UNIVERSITY OF LONDON THESIS

Degree PhD Year 2008 Name of Author ELLUL, Claire Denise

Functionality and Performance – Two Important Considerations when Implementing Topology in 3D

Claire Ellul

Thesis submitted for the Degree of
Doctor of Philosophy (PhD)
University of London

March, 2007

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Acknowledgements

It is absolutely impossible to mention everyone who contributed directly or indirectly to this work. However, the following deserve special mention.

My supervisors, Dr Mordechai (Muki) Haklay, Professor Peter Woodsford and Dr Tim Bevan, who have supported me throughout the research and writing process, providing invaluable advice, guidance and feedback at all stages of the project. Numerous fruitful discussions and their endless patience have both been of extreme value.

My colleagues and fellow students in the Department of Geomatic Engineering, who have often served as a sounding-board to discuss ideas and have also provided invaluable friendship and support throughout. Special mention goes to Jose Paulo de Almeida, Samson Ayugi and Katerina Christopoulou for both technical discussions and moral support. The support of the administrative and IT staff within the department has also been invaluable.

Dr David Chapman and Dr Roderic Bera have been a source of advice, encouraging me to explore areas of research that I would not have otherwise encountered. Dr Roland Billen at the University of Liege has proved an invaluable source of information and documents. Dr Sisi Zlatanova (Delft University of Technology) has kindly granted permission for the digital capture of the 3D 9-*Intersection* relationships diagrams published in her PhD thesis, and the Ordnance Survey licensed the use of their MasterMap ™ data for research purposes.

I would also like to thank my sponsors, the Engineering and Physical Sciences Research Council and 1Spatial Ltd. (formerly Laser-Scan, Cambridge), whose financial support has made this work possible. The Centre for Scientific Enterprise (London) also provided additional support for the investigation into commercial issues relating to this project.

Last but most definitely not least I would like to thank my family and friends, both in the UK and overseas. They encouraged me to embark upon this project and have supported me throughout.

Abstract

This thesis contributes to the understanding of the use of topology in analysing 3D spatial data, focussing in particular on two aspects of the problem – what binary topological analysis functionality is required in a commercial 3D Geographical Information System (GIS), and how should this functionality be implemented to achieve the most efficient query performance.

Topology is defined as the identification of spatial relationships between adjacent or neighbouring objects. The first stage of this research, a review of applications of topology, results in a generic list of requirements for topology in 3D. This was carried out in parallel with a review of topological frameworks and the relationships identified by one of the frameworks, Egenhofer and Herring's 9-Intersection, selected for implementation. Three generic binary relationship queries are identified (*Find Objects with a Specific Relationship*, *Find Intersecting Objects* and *What Relationship is there Between These Objects?)* and a mechanism described to allow these to be adapted to specific application terminology.

Approaches to the implementation of 3D binary topological queries include the use of data structures and an As-Required calculation, where computational geometry algorithms are run to determine relationships each time the user runs a query. The Three-Dimensional Formal Data Structure (3DFDS) was selected as a representative example of a Boundary-Representation (B-Rep) structure in GIS. Given the number of joins to be traversed when identifying binary relationships from a B-Rep structure, along with the requirement to query additional containment exception tables, an alternate structure, the Simplified Topological Structure (STS), was proposed to improve binary query performance.

Binary relationship queries were developed and comparative performance tests carried out against 3DFDS, STS and a Proxy for the As-Required calculation, using a 1.08 million object test dataset. Results show that STS provides a significant performance improvement over 3DFDS. No definitive conclusion could be drawn when comparing STS with the Proxy for the As-Required approach.

Table of Contents

List of Figures

List of Tables

Glossary

Term	*Definition*
2.5 Dimensions (2.5D)	Two-and-a half dimensions. In GIS, used in the representation of surface data, where each co-ordinate point (x, y) can be associated with at most one height value. This contracts with the true 3D case, where multiple height values can be associated with one (x,y) coordinate pair, allowing representation of vertical walls and cliffs not possible in the 2.5D case.
Boundary	A boundary point of A (where A is a subset of a topological space X) is any point p in X such that every neighbourhood of p has a non-empty intersection with both A and its complement X-A. The dimension of the space in which the sets are contained, known as the embedding space, must be considered when defining these components (Bruegger and Kuhn 1991). A line in two dimensional embedding space has no interior, but in one dimensional space it has an interior (the line excluding its end-points). The boundary of a disc in two dimensional embedding space separates its exterior from its interior. However, in three dimensional embedding space, the interior of a disc is in direct contact with the exterior.
Boundary Representation	A method for representing a 3D solid using its constituent surface components - Nodes, Edges and Faces. The interior of the solid is represented by the space enclosed by the surface. Nodes are used to define the Edges, which in turn are ordered to define each Face.
Closure	The closure of A (where a is a subset of topological space X) is the set of points p in X with the property that every neighbourhood of p has a non empty intersection with A. This is the smallest closed subset of X which contains A
Data Model	Frank (1992) defines a geometric data model as being used to describe a formalized abstract set of spatial objects classes and the operations performed on them.
Data Structure	Geometric data structure is the specific IMPLEMENTATION of a geometric data model, fixing storage structure, utilization and performance.
Database	"A collection of data used to represent information of interest to an information system" (Atzeni *et al*, 1998). A database can be paper based or computer based.
Database Management System (DBMS)	A software system used to manage collections of data. It provides functionality such as reporting, security and analysis to allow users to handle and process the information stored in a database. Examples of database management systems include bank transaction handling systems and airline booking systems.
Database Query	The process of extracting information from a database to answer a specific question – i.e. the process of interrogating the database.
Delaunay Triangulation	This is a triangulation of a series of points where the triangles are as nearly equilateral as possible (Worboys and Duckham, 2004).
Exterior	The exterior of A is the set of all points in topological space X that do not form part of A, where A is a subset of the topological space.
Geographical Information System (GIS)	A GIS is a computer-based information system that enables capture, modelling, manipulation, retrieval, analysis and presentation of geographically referenced data (Worboys 1995).
Homeomorphic	A homeomorphic transformation between two objects (also known as a continuous transformation) is a stretching and deformation taking one object to the other without cutting or gluing. Topological properties of an object are those that do not change under such a transformation.
Interior	A is a subset of a topological space X. A point p in A is called an interior point in A if p has a neighbourhood U in X that is contained in A. The interior of A is the largest open subset of A
Manifold	A sub-space M of the Euclidean space E^n is called a d-manifold ($d \leq n$) if and only if every point p of M has a neighbourhood in M homeomorphic to the unit d-dimensional ball (DeFloriani 2005). In other words, in 3D, for the manifold to be valid, the neighbourhood of each point within the sub-space must be able to be deformed into a sphere (a 3-dimensional unit ball). Hoffman (1989) gives a more specific definition for a manifold surface as one for which every neighbourhood of a point can be deformed into a plane. Thus self-intersecting surfaces are non-manifold.

A subspace M of the Euclidean space E^n is called a d-manifold with boundary ($d \leq n$) if and only if every point p of M has a neighbourhood homeomorphic either to the unit d-ball or to a half-ball. This allows for points on the boundary of the manifold. In a 3D case, the half ball is the unit 3D ball but with values of $d \geq 0$.

Nine-Intersections Framework

Defined by Egenhofer and Herring in 1990. Represents the topological relationship between two objects by examining the intersection of their interior, boundary and exterior. The relationship is represented in the form of a matrix:

$$R(A,B) = \begin{pmatrix} Int(A) \cap Int(B) & Int(A) \cap Bnd(B) & Int(A) \cap Ext(B) \\ Bnd(A) \cap Int(B) & Bnd(A) \cap Bnd(B) & Bnd(A) \cap Ext(B) \\ Ext(A) \cap Int(B) & Ext(A) \cap Bnd(B) & Ext(A) \cap Ext(B) \end{pmatrix}$$

Non-Manifold

See Manifold

Relational Database

A type of database where the data structure consists of multiple tables which can be linked to each other to provide a full information picture. For example, an airline booking database could contain tables for airline details, passenger details, flight details and airport details. These four tables can then be linked to form a complete picture of the process required to book a flight.

Simple Polygon

Topologically equivalent (homeomorphic) to a circle – i.e. can be deformed into a circle without tearing, cutting or gluing. Simple polygons do not have internal holes and are not composed of multiple disconnected parts.

Simple Polyhedron

Topologically equivalent (homeomorphic) to the sphere (can be deformed into a sphere without tearing, cutting or gluing). Simple polyhedra do not have internal cavities and are not composed of multiple disconnected parts.

Simplex

An n-simplex can be defined as the simplest bounded region in n-dimensional space. Basic simplices are given as follows:

- A node is a 0-dimensional simplex
- An edge is a 1-dimensional simplex
- A triangle is a 2-dimensional simplex
- A tetrahedron is a 3-dimensional simplex

Structured Query Language (SQL)

The language used to query (ask questions of) a relational database.

Three Dimensional (3D)

Refers to any data for which every (x,y) coordinate can be associated with MORE THAN ONE height value.

Topological Space

DeFloriani (2003) defines a topological space as a pair (X,T) where X is a set and T is a topology for X. Thus a topological space is defined by specifying the collection of its OPEN sets. Open and closed sets for the unit circle are defined as:

- Open set – $C = \{(x,y) | x^2 + y^2 < 1\}$
- Closed set – $\bar{C} = \{(x,y) | x^2 + y^2 \leq 1\}$

Topology

DeFloriani (2003) defines a topology for a set X is a family T of subsets of X satisfying the following properties

- X and the empty set Ø belong to T
- The union of any collection of elements of T is in T
- The intersection of any finite collection of elements of T is in T.

Any set of subsets of S obeying these rules is thus a topology on S. All the subsets in this set of subsets are open. The complement within S (i.e. the parent set) of any open set is a closed set.

Voronoi

The Voronoi diagram is the dual of a Delaunay triangulation (Worboys and Duckham 2004), where in vertices in the Delaunay triangulation become centre-points in the Voronoi diagram. Each region in the diagram is created such that every point in that region is closer to the centre point than to any other centre point. This property allows the application of Voronoi diagrams and Delaunay triangulations to problems such as the creation of catchment areas for schools, fire stations, ambulance services and so forth.

1 Introduction

The manipulation of spatial data is progressively more important in today's information-driven world. Spatial data is involved in a wide range of applications, ranging from decision support through planning and design to medicine and robotics; increasingly this data is three-dimensional (3D). This thesis contributes to the study of the use of topology in analysing 3D spatial data, focussing in particular on two aspects of the problem – what topological functionality is required in a 3D Geographical Information System (GIS), and how the foundation of this functionality, binary relationship identification, can best be implemented for query performance.

This introductory Chapter of the document presents an overview of GIS in 2D and 3D and highlights the importance of topology in this context. Issues relating to 3D topology are reviewed, and research questions derived to address them. The Chapter concludes with an overview of the structure of the remainder of this document.

1.1 2D GIS

A GIS is defined as a "computer-based information system that enables capture, modelling, storage, retrieval, sharing, manipulation, analysis, and presentation of geographically referenced data" (Worboys and Duckham 2004). GIS can be used in fields as diverse as archaeology, utilities management and health.

Traditionally GIS and the paper maps from which they originate are two-dimensional. 2D GIS utilises the concept of themed layers to model data. Map layers, generated from data (which can be stored in an object-relational database for integration with non-spatial data) are stacked (overlaid) to build up a representation of the 3D world. Each map layer contains one object theme, which can in turn represent physical features such as parks, buildings, lakes, roads or railway lines, or can represent other concepts such as administrative boundaries or demographic information (such as the ethnic mix of an area). Within each layer, data can be represented as points, lines or polygons or a mixture of these – each of these are made up of one or more two-dimensional coordinates (x, y) representing the position of the feature. Aerial photography or satellite images can also be added.

This data can be used to underpin analysis – for example, shortest path algorithms can be run against line data representing road centrelines to find a route between two points, demographic data can be analysed to provide support for the location of a new supermarket.

Although 2D GIS are well established, the world is three-dimensional and the third dimension is becoming an increasingly important element of the decision support process.

1.2 3D GIS

To underpin day-to-day activities of users ranging from geologists, earth scientists and petrochemical engineers to architects, urban planners, utility companies and local and national government, information systems must be able to respond to questions that can more easily be answered in 3D. Examples of such questions include:

- "How far under 'High Street' does the gas main lie?" – in 2D GIS coordinates do not store height information and depth is stored as an attribute of the gas main. It is often not clear from which point the measurement has been made – from the centre of the pipe, the top or the bottom of the trench.
- "What rock strata surround the potential oil field and how far down does the drill have to go?" – 2D GIS cannot easily model the concept of a vertical drilling process – the 2D stacked layer approach may contain descriptions of the various types of rock formation but cannot generate a 3D depth picture.
- "I want to create an underground parking area - who has rights to the land underneath the Local Council Offices?" – presents a similar issue to the one above – 2D GIS cannot analyse the complexities of land ownership and rights – the concepts of above and below cannot be expressed, and ownership may also change depending on the depth below ground. Such queries are particularly relevant for utility and underground transportation.
- "Will this wind farm impact wildlife in the area?" - such impact needs to be monitored at different heights above ground, to determine impact on ground-based animals and birds. The results of such monitoring cannot easily be visualised or analysed in 2D GIS, which does not allow values to be stored at multiple height values for the same ground coordinates.
- "What is the shortest evacuation route from my current location in this building?" - 2D GIS in unable to model the complexities of a 3D building, including lift shafts, stairwells, fire escapes and fire doors and the relationships between these features (is it possible to get to the exit through this stairwell).

These questions exemplify the problems that GIS users are required to solve on a regular basis to support decision making processes. Incorrect information provided due to the limitations of the 2D systems described could lead to potentially dangerous situations, have negative financial implications or cause long-term environmental impact. 2.5D GIS (which allows the association of one height value with each coordinate pair) does not provide an adequate solution to these problems, which require multiple height values to be represented for the same coordinate pair to facilitate the construction of true 3D objects.

3D GIS allow users to easily and intuitively understand geometric data, supporting the decision making processes in various areas of specialisation. As Frank and Kuhn (1986) state - "geometric objects[1] and properties are not directly observable but are a product of an abstraction process applied while observing and describing reality". The use of three dimensions provides a better abstraction due to the creation of volumetric objects, which more closely represent the real-world.

1.2.1 Contrasting 2D and 3D GIS

The elements of a GIS listed by Worboys and Duckham (2004) in a 2D context (database, data processing, data storage and retrieval, data sharing, data presentation, spatial reasoning, spatio-temporal analysis) are directly transferable to 3D GIS. However, a number of key differences can also be identified:

- As described above, 3D GIS can model and analyse situations which cannot be handled in a 2D system.
- 3D GIS require additional data types beyond point, line and polygon. Polyhedra (representing closed 3D volume objects) should also be modelled.
- Data quality issues are additionally complex in 3D – whereas in 2D all data is automatically planar, this is not the case in 3D. Quality control processes must also validate that polyhedra are closed (see Appendix 1 for examples of quality control algorithms). Data errors in 2D include slivers (where data overlaps), undershoots (where a polygon is not closed correctly) and overshoots (where a line extends past the intended boundary). The additional dimension in 3D complicates the resolution of these problems.
- The additional height value for each coordinate point increases the volume of data to be stored. The additional coordinate points created in 3D (for example to represent a building as a 3D polyhedron rather than a 2D polygon) add to the complexity.
- The algorithms required for data analysis and visualisation are additionally complex - for example, a new requirement for volume calculation can be identified. Section 4.3 provides details of this issue with respect to the determination of topological relationships).

1.2.2 Advances in 3D GIS

An increasing number of true 3D (i.e. with multiple height values for each coordinate pair location) datasets are becoming available as hardware and software capabilities are augmented to support their efficient capture, maintenance and manipulation. This is evidenced by the number of applications supporting visualisation of such data (a number of these are listed in Section 3.1). Designers have also been creating 3D objects with computer-aided design tools

[1] Throughout this thesis, the term *feature* is used in the context given by Pilouk (1996, pg. 93) or by ISO 19107 (OGC 2006), and represents both thematic information (including attributes such as ID) and geometry. The term *object* refers to the geometry of a feature (known as a spatial object in ISO 19107).

for over 25 years. The Ordnance Survey (Ordnance Survey 2004) is engaged in research to examine 3D representation of topographic data, including street furniture, deriving this information from aerial photography via the process of photogrammetry. Other sources of 3D data include datasets developed using the CityGML standard (CityGML 2006) and 3D imagery such as that obtained from hand-held laser-scanners. It is also possible to construct 3D data from LiDAR (Light Detection and Ranging) data or aerial photography. Products such as SketchUp (Google 2007) provide a 3D modelling tool aimed at non-specialist home users, and allow them to upload data to a common repository for access by other users, although the quality of the resulting 3D model does depend on the expertise of the authors.

Relational and Object-Relational databases, traditionally associated with non-spatial data storage or with 2D GIS, are also being enhanced to handle 3D data (Oracle 2007, Informix 2007 pg. 49, PostGIS 2007 pg. 15). The advantages of database use are many, and are equally applicable to 3D GIS and other information systems contexts. Using a database, the tools offered (such as visualisation of the data and data analysis) can be integrated rather than taken from a combination of piecemeal applications developed for a specific purpose. 3D data stored in the database can be integrated with other datasets, allowing spatial and other organizational data to be analysed concurrently. Centralised management of data and tools ensures that duplicate copies are not made and allows a consistent view of the information to be presented to all users. Databases can also be scaled up to allow large numbers of users to access data simultaneously and they provide multiple levels of security. They are designed to handle large volumes of data, such as an urban model for an entire city. The Structured Query Language (SQL) toolkit provided with many databases allows a standardised cross-platform approach to data creation, maintenance and interrogation, no matter which underlying database software is chosen.

1.2.3 Shortcomings of 3D GIS

Despite the progress in 3D GIS described above, a fully-functional 3D GIS, supporting the equivalent of all functionality provided in 2D (including data capture, presentation and analysis), has not yet been developed commercially. Pfund (2001) notes that the impediments to the development of 3D GIS include the difficult technical and conceptual problems that need to be addressed and the high implementation costs, although the latter have now decreased. A brief review of existing 3D GIS systems and users reveals that, five years on, a number of specific issues still remain to be resolved before an implementation of 3D GIS can take place. These can be grouped into two areas – namely data and software.

Considering data first, a review of the sample 3D questions listed above reveals that they cannot be answered by a simple visual analysis of the relevant 3D geometry. As with 2D GIS, decision support in 3D requires the integration of multiple datasets – both spatial and non-spatial - into a single environment. This allows them to be cross-referenced for analysis purposes. However, much true 3D data is currently held in Computer Aided Design (CAD) format (although true 3D data support will be included with Oracle 11g, Oracle 2007), in specialist petrochemical applications or in proprietary software formats. Thus, despite the opportunities offered by Relational and Object-Relational databases, they cannot easily be integrated into a corporate system for decision support purposes. Additionally, 3D GIS tend to be department-based and specialized to a particular application. Due to this, 3D data is typically held in isolation of corporate data. Data integration and exchange is also limited due to differing conceptual approaches to modelling the data.

Examining GIS software, and in particular commercial software, it can be seen that some implementations of 3D GIS do exist (for example van Oosterom *et al.* 1994) and some specific algorithms have been designed (tetrahedral overlay, van der Most 2004). However, generic extensions to the SQL query language to support volume calculation, distance measurement in 3D and other analysis including topology, has not been defined or implemented. Many 3D and 2.5D GIS, whether underpinned by a database or otherwise, focus on visual analysis of the data (Ellul and Haklay 2006) and lack functionality to support other forms of querying, such as metric and topological analysis. Many GIS claiming to be 3D in fact only support 2.5D analysis, although the latter cannot handle truly vertical walls, overhangs or steep cliffs due to issues with vertical Faces (De La Losa and Cervelle, 1999). Zlatanova *et al.* (2002) concur that true 3D GIS are as yet undeveloped, noting that they consist of combinations of Computer Aided Design (CAD), Database Management Systems (DBMS) and 2D GIS systems. This leads to problems of software integration - users must query two different systems to obtain the answers they require.

The following issues must therefore be addressed to further 3D GIS implementation and thus provide users with a fully functional 3D GIS:

- Proprietary data formats and software prevent integration of data from diverse sources, and incorporation of data into corporate databases.
- Improvements to 3D GIS software, including the ability to handle true 3D data and perform analysis beyond visualization, are also required.

Efforts are currently being made by the Open Geospatial Consortium (OGC, 2006) towards data interoperability and open data exchange standards such as the Geographic Mark-up Language

(GML). Further work towards standards for exchange of 3D urban data is being carried out by the Special Interest Group 3D (SIG 3D) of the Geodata Infrastructure of North Rhine-Westphalia (GDI-NRW), who noted that many cities are constructing 3D city models for applications including urban planning, disaster management, tourism, vehicle and pedestrian navigation and facility management. Given that no standard for the exchange and integration of such data exists, they propose CityGML (CityGML 2006) for this purpose.

Both of these standards are applicable in the 3D context. The research described in this thesis therefore focuses on the second of the issues listed above – namely the requirement for improved analytical functionality.

1.3 The Importance of Topology in GIS

Brown (1988) notes the importance of topology to science in general, as it gives a precise but general sense to the intuitive ideas of nearness and continuity. The first use of topology as a tool has been attributed to Euler, who proved that the Konigsberg Bridge Problem is insoluble in 1735 (Worboys 1995). Since then, topology has evolved as a branch of mathematics, and more recently as a central core feature of GIS (Theobald 2001).

Topology underpins areas of GIS functionality including object overlay, object relationship identification and network analysis. The term topology refers to "the aspect of GIS which allows specialized analytical operations of spatial search and overlay. It refers to the spatial relationships between the points and lines that define the geographic features" (Montgomery and Schuch 1993) and examines relationships between places and spaces from the point of view of their position relative to other places and spaces (Laurini and Thomson, 1992). DeMers (1997) defines it as the identification of spatial relationships between adjacent or neighbouring objects and states that it is used to allow specialist analysis that focuses on the relationships between objects such as buildings, archaeological finds and geological strata. McDonnell and Kemp (1995) identify topological properties including adjacency and containment.

Thus, the term topology refers to the relationships that can be determined from the positional information of one object in relation to another. In a 3D context, topology answers questions including "which artefacts can be found within a particular layer of stratigraphy?", "which model regions are cut by a particular fault?", "find any water pipes that intersect with the trench to be dug in this road" and "find the shortest escape route through this 3D building". Two types of topological relationship can be identified in GIS – binary (i.e. between two objects) and higher order (between multiple objects). The latter can be determined from the former – for example, if Road A links to Road B (a binary relationship) and Road B is linked to Road C

(another binary relationship) it is possible to walk from Road A to Road C. Baars *et al.* (2004) note that storing all topological relationships between all objects of interest requires a high amount of storage space. Therefore it is more convenient to store relevant relationships (e.g. neighbours) and compute the others. Thus binary topological relationships form the foundation of higher order relationships and efficient determination of binary relationships will in turn lead to efficient determination of the relationships at higher order. Given that it is impossible to predict order as this will vary from query to query, the research described here will focus on binary relationships.

A number of commercial GIS software products, including ArcGIS (ESRI 2006), Geomedia Professional (Intergraph 2006) and Smallworld (GE Energy 2006) offer 2D topological functionality. Additionally, two server-side topology products can be identified in the context of Object-Relational databases. Oracle 10g Topology supports network and planar topology management for 2D data, in the form of topology structures separate from the standard spatial data type. Radius Topology, developed by 1Spatial (Laser-Scan 2007), offers 2 and 2.5D topological functionality in conjunction with Oracle Spatial again deals with both network and planar topology. However, none of these products currently offer topological functionality in a true 3D setting – i.e. with the ability to handle topology of closed 3D objects and multiple height values at one point rather than surfaces having single height values at a single point. The lack of topology in 3D GIS motivates this research.

1.4 Research Issues in Topology

Despite topology being well-established in 2D GIS, forming part of most software packages, issues in relation to topology in 3D GIS are still the subject of ongoing research. A number of these are listed here.

1.4.1 Issue 1 –User Requirements for 3D Topology

In order to implement 3D topology, it is important to have a clear understanding of how end-users utilise and understand topology. One feature common to many implementations of 3D topology is their specificity – the solutions provided have, in general, been designed to solve well-defined problems such as internet visualisation of 3D data (Zlatanova 2000), support for urban planning (Coors, 2003) or geological applications (Gong *et al.* 2004). The functionality provided with each implementation, and the range of queries implemented for the user, is generally focussed on the tasks that need to be accomplished within the selected domain. For example, the implementation of the Simplified Spatial Model (SSM) by Zlatanova (2000) was designed to support web visualisation of urban 3D data and thus the interface only supports simple queries identifying shared primitives between the 3D objects. Zeitouni *et al.* (1995)

describe an implementation of the 3D relationship concepts of "above" and "below", due to the required support for urban modelling. However, 2D GIS is utilised in a wide range of disciplines, and it can be assumed that a similar situation will occur for 3D GIS – therefore cross-disciplinary requirements should be identified through a review of existing uses of 3D spatial data.

1.4.2 Issue 2 – Mapping End-User Relationships to Theoretical Topology

Figure 1 below illustrates this issue in the context of one type of topological relationship between objects, namely the definition of the term adjacency.

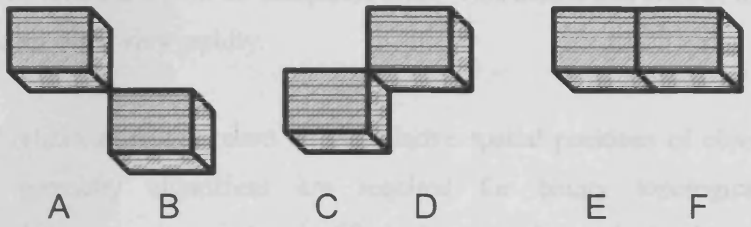


Figure 1 - Alternate Adjacency Relationships

Objects A and B show point adjacency for two 3D blocks, C and D show line adjacency and E and F show surface adjacency. Defining adjacency is additionally complicated when it is considered that not all 3D objects are box shaped – they may be surfaces, lines, complex 3D bodies and so forth. Interpretations of the word adjacency may also be application or domain-specific. Therefore, although terminology may be well understood by the end-user, the understanding may differ between users and application domains.

To support computer implementation a more precise definition of each relationship is required. Topological frameworks (see Chapter 2) provide a theoretical means to more formally describe relationships between objects, giving a comprehensive method of differentiating between the various relationships. They also provide the groundwork for implementing topological modelling in a computer-based system, as a series of well-understood rules is required to define and identify relationships between objects. However, the number of relationships identified by these frameworks is high when compared to those understandable by non-specialised user. For example, a total of 143 relationships between simple objects have been identified by Zlatanova (2000) for the Egenhofer and Herring (1990) 9-Intersection Framework in three dimensions.

A mechanism is therefore required to map a large number of framework relationships to those identified by end-users, and to ensure that this mapping is flexible enough to accommodate different application domains.

1.4.3 Issue 3 – Efficient Determination of Binary Topological Relationships

The emergence of increasingly larger 3D datasets, particularly in the context of the representation of urban environments, puts additional load on computer systems. Many iterations of binary queries may be required to provide the decision support information required by end-users. For example, to determine if any optical fibre cables intersect a proposed trench in a road, the intersection relationship between each cable and the trench must be determined. As the fibre cable dataset for a city is typically very large, the number of binary relationships to be identified could be quite high. Binary queries also form the foundation for networking functionality. To provide acceptable performance to users, answers to such questions must be returned within an acceptable time. This means that each of these constituent queries must be executed very rapidly.

As topological relationships are based on the relative spatial positions of objects, a series of computational geometry algorithms are required for binary topological relationship determination. Two approaches can be identified when considering the implementation of these algorithms – As-Required calculation and the use of topological data structures. As-Required implementations utilise computational geometry to directly determine relationships each time they are queried by an end-user. Topological data structures are based on the principle of 'calculate the relationship once and query the result many times'. A topological engine makes use of computational geometry algorithms to determine any shared geometry – for example, the common area of wall between two buildings. The results of this calculation are then stored in a data structure, which this can be queried by any number of end-users.

No consensus has been achieved on the selection of the As-Required approach or a structure-based approach in the 2D case, with both approaches being available in mainstream GIS. Issues relating to the storage requirements for each approach also weigh into the debate – a trade-off may be required between relationship identification performance and storage. In 3D, no commercial As-Required implementations are yet available, although Oracle has stated that full 3D functionality will be incorporated in their next release, 11g (Oracle 2007). This is possibly as a result of the performance implications of the additional complexity that a third dimension adds to the computational geometry process. A review of structure-based approaches is given in Section 4.6.

Wei *et al.* (1998), Wei (1996) and Zeitouni *et al.* (1995) cite the lack of a standard approach to, and model for, topology in 3D GIS as being one of the major impediments to its uptake. Pfund (2001), however, suggests that topological information is usually omitted for performance

reasons. This view is supported by the complexity of the algorithms, and the increasingly larger datasets against which they must be run. Efficient binary relationship determination drives the selection of an implementation method.

1.4.4 Issue 4 – Integrating 3D Topology into 3D GIS

Binary topological queries are rarely executed in isolation of other GIS analysis such as metric or attribute-based queries, and end-users may not even be aware of the distinction between such queries. Users do not generally ask 'what topological relationship exists between A and B'. In fact, analysis generally combines the topological relationship with other spatial or attribute queries. In the example above (Section 1.4.3), this could include distance of the fibre cable below the ground, and the type of cable. A mechanism is therefore required to ensure that a seamless, integrated analytical interface is presented to end-users of 3D GIS.

Given that much enterprise GIS data is held in relational or Object-Relational databases, which are in turn queried by standardised Structured Query Language (SQL), current limitations of SQL are also relevant here. The standardised version of SQL does not support topological queries, and thus requires extension. This should be accomplished in such away as to support the integration of topology with other standard and non-standard SQL queries.

1.4.5 Issue 5 – Promoting Uptake of 3D GIS

An additional aspect impeding the uptake of 3D topology is lack of understanding of its potential. Once topology has been incorporated into 3D GIS, this problem can be redefined as that of promoting 3D GIS in general and hence the uptake of the incorporated 3D topological systems. This encompasses broader business issues including availability of data, identification of target market areas, prototype demonstration, lead user identification and potential revenue and competition evaluation. Given the lack of true 3D GIS, and hence the lack of market sizing information, the commercial feasibility of 3D topological functionality is also unknown at this point.

1.5 Defining the Research Questions

The issues described can be broadly categorised into two areas - functional requirements of a cross-disciplinary topological implementation (what should 3D topology do?) and the optimal method to approach implementation to ensure that the resulting performance is acceptable (how should it do what is required?). This research sets out to answer these questions, providing information to support the move from domain-specific, isolated, applications of 3D topology towards a generic implementation to be incorporated into a 3D GIS. In a wider context, it also

examines how such functionality should be developed and packaged to appeal to a broad user base.

More formally, the research questions have been identified as follows:

- Which areas of GIS applications will benefit most from efficient handling of topology in 3D? Which binary topological relationships should be modelled to generate this benefit?
- How can these relationships be implemented in an Object-Relational database context?
- Of the various implementation options, which approach provides the most efficient performance results for binary relationship queries?
- What additional considerations should be made to support the inclusion of 3D topological functionality in 3D GIS?

1.6 Overview of the Research

To answer the research questions listed above questions, a number of steps were identified. These are described here.

1.6.1 Identify Requirements and Map to Corresponding SQL Queries

Firstly, a comprehensive review of literature relating to users and applications of 3D data was carried out (this is described in Chapter 3). The aim of this review was to identify analytical tasks that are underpinned by topological queries - i.e. where information relating to adjacency or containment of objects is required to perform the analysis. As many users of 3D data are not consciously aware that the nature of their analysis is topological, and familiarity with the potential of 3D topology in general is low, this was not deemed to be sufficient to identify all requirements. Therefore, the second part of this process involved a review of current implementations of topology in 2D and 2.5D GIS, using literature and case-studies from GIS vendors.

The results of each part of the review process were combined to generate a set of functional requirements to be met by a 3D topological system. These were then grouped into general cross-disciplinary requirements and those that are specific to a particular application. This allowed prioritisation of requirements and helped to focus subsequent development efforts. Requirements for binary topological queries, which form the main focus of this research, were also extracted and three generic query types identified in this context. These were defined as:

- Query Type 1 – Find Objects with a Specific Relationship to this one.
- Query Type 2 – Find Objects with Any Non-Disjoint Relationship to this one.
- Query Type 3 – What Relationship is there between Object A and Object B?

In parallel with the requirements gathering process, a review of topological frameworks was undertaken (see Chapter 2). Given the overall aim of this project, i.e. to move towards a generic implementation of 3D Topology, existing standards were also examined in order to select an appropriate framework for implementation. In particular, frameworks identified by the Open Geospatial Consortium were considered. The 9-Intersection Framework developed by Egenhofer and Herring (1990) was selected for implementation due to its inclusion in the OGC standard and the availability of additional research on the framework within the context of complex objects and 3D binary relationships.

The many relationships identified by the selected framework do not directly correspond to the terminology identified as a part of the requirements gathering process. Therefore, a mechanism for mapping the relationships identified by the frameworks to those identified by the requirements gathering process was designed, encoding the 9-Intersection Framework relationships numerically and mapping these numbers to terminology selected by end-users. Flexibility is important here, as multiple application domains utilise 3D GIS functionality.

1.6.2 Identify Approaches to 3D Binary Relationship Identification

Once the user requirements and corresponding SQL queries were identified they formed one factor feeding into the design for a system to query topological relationships in 3D data. Two approaches were reviewed and compared – As-Required (where relationships are computed each time the user runs a query, see Section 4.4.1) and structure-based (where relationships are pre-determined and the results stored in a data structure, see Section 4.4.2). A review of the dominant structure-based approach in 3D (Boundary-Representation) in the form of the Three-Dimensional Formal Data Structure (3DFDS, Molenaar 1990) identified two issues that may impact performance – the number of relational joins that must be followed to identify the interior and boundary primitives of each object and the inclusion of additional exception tables to model containment situations. An alternative structure, the Simplified Topological Structure (STS), was designed and created as part of this research, with the aim of improving binary query performance. STS is described in Chapter 5.

Although theoretically plausible (as join queries are costly within a relational database query, Atzeni *et al.* 1999) the query performance improvements offered by STS required validation through performance testing. Additionally, research did not reveal any concrete evidence that the use of a data structure provided improved performance when compared to an As-Required determination of the 9-Intersection relationships.

A 1.08 million object test dataset, binary query algorithms and comparison tests were thus designed to validate that STS does provide improved performance when compared to 3DFDS, and to investigate issues relating to structure-based implementation versus As-Required calculations.

1.6.3 Developing a Dataset to Compare As-Required, STS and 3DFDS

The test dataset (see Chapter 6) developed to support the performance testing process included a comprehensive set of objects representing the 3D topological relationships between simple objects identified by the 9-Intersection framework (based on work carried out by Zlatanova 2000). To support realistic performance testing, this test dataset was replicated (multiplied) to a total of 1.08 million objects.

Although comparative performance testing was the main focus of the research, the selected test dataset additionally validated that relationships identified to support the end-user requirements (i.e. those defined by the 9-Intersection framework) can be implemented by the structures. The dataset was also used to compare storage requirements for the selected implementation approaches (see Section 6.4.2).

1.6.4 Algorithms to Compare As-Required, STS and 3DFDS Approaches

Set-based algorithms were developed to implement the 9-Intersection queries against the three selected structures. Due to the set-based, declarative nature of SQL and the complexity of the process to determine 9-Intersection relationships, PL/SQL, a procedure-based extension which converts SQL to a programming language, was used for query implementation (see Section 7.7.1).

The development of a topological engine for an As-Required implementation falls outside the scope of this research and has not been undertaken commercially. A Proxy, suitable as a representation of the time taken to determine 9-Intersection relationships using computational geometry algorithms, was thus used. As the first part of the As-Required query process involves filtering, and a 3D spatial filter process is available, it was determined that this could be used as an initial Proxy for the computational geometry algorithm. This was improved by incorporating the results of 2D binary topological queries (see Section 8.4.4).

1.6.5 Tests to Identify the Most Efficient Implementation

Three data structures were selected for performance testing - the Proxy for the As-Required queries, the STS and the 3DFDS (Molenaar 1990). Each structure was populated with the replicated test dataset described above, in an Object-Relational database. A series of multi-user

tests were implemented to test scalability for the selected structures, with tests run for a single user then for 2, 4, 6 and 8 users concurrently (scalability testing). Four additional subsets of the dataset were also created, ranging in size from 264 Objects to 135,168 Objects. This, in conjunction with the 1.08 million Object dataset, allowed the behaviour of each approach to be measured in terms of increasing data volume (algorithm complexity testing).

For each dataset size/concurrent user number combination, performance was evaluated in terms of each of the three queries identified from the user requirements review – i.e. *Find Objects with a Specific Relationship*, *Find Intersecting Objects* and *What Relationship is there between A and B*? Test design is described in Chapter 8, with results being presented in Chapter 9.

To conclude the analysis, a review was conducted in relation to the commercial applications arising from this research, to identify additional development work and testing required (see Chapter 10). Along with this, the commercial potential of 3D topology was evaluated in the context of the 2D GIS market. Likely partners were identified, and favourable trends and threats reviewed.

1.7 Research Outcome

In summary, this research shows that:

- A requirement for topological analysis of 3D data does exist and that this functionality would enhance 3D GIS.
- It is possible to derive a list of cross-disciplinary requirements through a review of literature related to applications utilising 3D data and current uses of 2D topology.
- Three generic of topological relationships (derived from the requirements) can be identified and mapped to the relationships identified by topological frameworks such as the 9-Intersection framework. Terminology/relationship cross-referencing can be implemented in such a way as to allow users the flexibility to identify the meaning of each relationship on a domain-specific basis.
- It is possible to develop and replicate a test dataset to illustrate the 9-Intersection relationships between simple 3D Objects.
- Set-based algorithms can be used to determine binary topological relationships between Objects in the context of the *9-Intersection* Framework.
- Comparing the structure-based approaches, STS requires 12% less storage than 3DFDS (for our test set of 1.08 million Objects). Executing binary relationship queries, STS again out-performs 3DFDS by a factor of approximately 10 (depending on the query type).

- Comparing STS with a Proxy for the As-Required approach is less conclusive. STS again demonstrates generally better results in terms of query performance but further research is required due to the nature of the selected Proxy. The As-Required approach does not utilise the additional storage needed for a data-structure approach.
- An integrated GIS vendor, who supplies services from data manipulation to consultancy, would be best placed to generate value from the 3D Topological Toolkit resulting from this research.

Overall, the outcome of this research therefore provides information to support the process of implementing topological functionality within 3D GIS. This is of further value when it is considered that the approaches described can be implemented within the context of any Object-Relational environment implementing spatial object types.

1.8 Project Scope

A number of pre-existing factors influence the direction taken by this research, key amongst these being the involvement of Laser-Scan as commercial sponsor. Laser-Scan (now known as 1Spatial) is based in Cambridge, United Kingdom, and was founded in 1969 by three academics from the Cavendish Laboratories (the Physics department of the University of Cambridge), initially to develop and sell a machine using a laser beam to follow lines on photographs. This led to the development of technology to support Digital Mapping in 1975. This was initially hardware-based but map editing software was added in the late 1970s. The company expanded from map production into GIS in the 1980s with the release of XGIS.

As part of their current portfolio, Laser-Scan develops technology integrated with large spatial databases, providing tools to validate, clean and merge spatial datasets. The choice of software tools and development environment selected for this research has been driven in part by their existing software products, in particular Radius Topology, which provides 2.5D topological functionality, including both data quality management and analytical querying, for spatial data held within an Oracle database. In line with the existing product suite, the topological analysis functionality developed as part of this research has thus been implemented within an Oracle database, although any modelling and structure design undertaken is platform-independent.

Additional scoping decisions resulting from Laser-Scan's sponsorship are given here. These served to define the boundary of the research and to ensure that the research outcome not only satisfies the requirements of the PhD process but is also relevant to the commercial sponsors of this project.

- In a similar manner, the 3D geometry storage model itself can be ignored as this will be taken to be the existing Oracle Spatial SDO_GEOMETRY model. No investigation

into the different options for modelling 3D geometry is required unless this impacts directly on the topological model.

- The project deals with true three dimensional spatial data, and handling 2.5D data and time series data is also outside project scope. Similarly, only discrete phenomena are to be considered at this stage.
- The development of data capture, structure population and data cleaning processes relating to 3D data (i.e. the topology engine) will not be considered as part of the project, apart from where this process relates to potential applications of topology or may impact structure design. Functionality to automate this process may be developed by Laser-Scan and is therefore commercially confidential.
- The work focuses on a database implementation of topology. This should be designed to be accessed via SQL. The development of a front-end application to access the functionality developed is outside the scope of this research.

1.9 End-Users of this Research

The potential end-users of this research fall into three groups. The first group those requiring topological analysis within the context of a 3D GIS. This includes users of 3D data and existing proprietary systems, as well as users of 2D GIS whose needs may be better addressed in 3D. Amongst these are local and national governmental organisations, urban planners, architects, archaeologists, geologists, and petrochemical scientists. The second group comprises developers and vendors of 3D GIS software working with Object-Relational databases. The results of this research may, in fact, guide product development for this group. Finally, academic researchers in the field of topology may also find the results described here useful as a foundation for further work or in complement to their own research.

1.10 The Structure of this Document

A number of theoretical and practical stages formed part of this research. These are illustrated in Figure 2 below, which also identifies corresponding Chapters in the document. The upper part of the Figure is repeated through the document to give each Chapter context in terms of the overall research process.

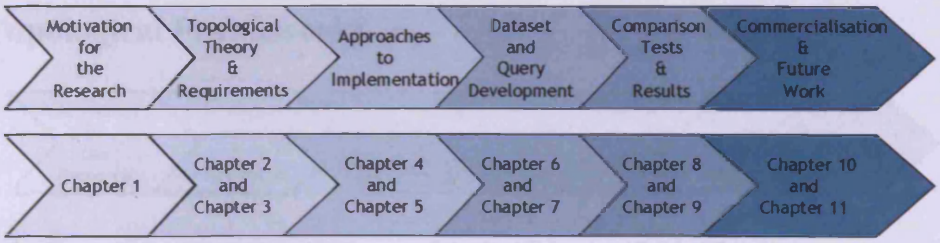


Figure 2 - Overview of Document Structure

Chapter 2 introduces the theory behind topology in a GIS context and reviews topological frameworks. Chapter 3 presents the requirements gathering process, detailing application areas reviewed and listing the resulting requirements for topological functionality in 3D, along with the selected groupings and prioritisation. The selected framework is cross-referenced to the requirements and a list of required topological queries is generated.

Chapter 4 presents an overview of current and potential implementations of topology within a 3D context, detailing how topological relationships are derived from 3D data, either using computational geometry directly or through the use of data structures. A review of existing data structures is described and reveals that a relatively high number of relational joins must be followed, and additional tables queried, to identify the topological relationships required. Chapter 5 therefore describes an additional structure, designed to optimise performance of binary topological queries – the Simplified Topological Structure.

Three approaches, two structure-based and one using a Proxy for the direct computational geometry algorithms, are taken forward for performance testing, and Chapter 6 describes the population of the three data structures within an Object-Relational database, along with the process of generating and replicating a test dataset for each of the structures. Chapter 7 gives an overview of the set-based algorithms designed to implement the selected queries.

Chapter 8 describes the performance tests carried out against each of the three implementations, with Chapter 9 describing the results analysis process, identifying the most efficient approach for the rapid querying of 3D topological relationships within an Object-Relational database.

Taking into consideration the research and results described, Chapter 10 defines a product, namely the 3D Topological Toolkit, and then describes considerations relating to the commercialisation of this product. Finally, Chapter 11 concludes the thesis with a review of the work undertaken and results obtained. The research questions defined in this Chapter are reconsidered, and further work suggested.

2 Topological Frameworks

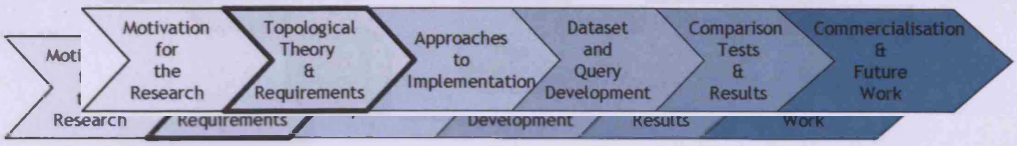


Figure 3 - Overview of Document Structure showing Context of this Chapter

2.1 Introduction

Topology has been called the "general study of continuity" (Frank and Kuhn 1986). This Chapter introduces topology as understood a GIS context, focusing in particular on binary topological relationships. This is followed by a review of frameworks defining such relationships and the selection of a single framework, to underpin further research. Framework selection is based on three criteria – existing standards, the applicability of the framework in a 3D context and the ability of the framework to handle real world objects (such as multi-part objects and those with internal holes or cavities).

2.2 Topology and GIS

Montgomery and Schuch (1993) define topology in the context of GIS as:

> *"The aspect of GIS which allows specialized analytical operations of spatial search and overlay. It refers to the spatial relationships between the points and lines that define the geographic features".*

Similarly, Laurini and Thomson (1992) state that topology examines relationships between places and spaces from the point of view of their position relative to other places and spaces.

These definitions relate to the aspect of topology relevant to GIS of greatest interest to this research – i.e. topological relationships between objects. However, other aspects of topology can also be identified as relevant. Work by Corbett (1979) and White (1984) relates the mathematics of graph theory and the simplexes and complexes described in the context of algebraic topology to the storage of geographic data. This underpins a number of data structures utilised in modern GIS, examples of which are given in Chapter 4. Similarly, applying the concept of manifolds to these structures results in a mechanism to automatically validate the geometry (coordinates) of stored objects according to pre-defined rules as part of the structure population process. Examples of such rules are given in Appendix 1.

2.2.1 Topological Relationships

The binary relationship terms adjacency and containment further exploration. Figure 4 below illustrates one issue in the context of adjacency.

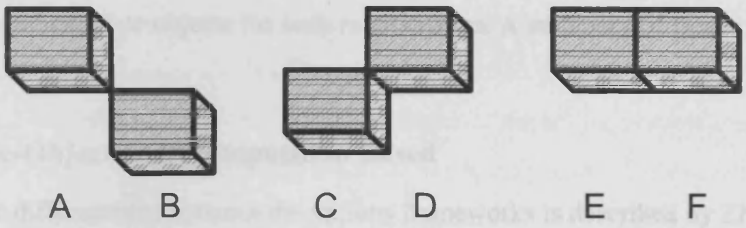


Figure 4 - Alternate Adjacency Relationships

As can be seen from Figure 4, there is no one clear interpretation of the word adjacency. Objects A and B show adjacency as two 3D blocks sharing a point, C and D show a shared linear object and E and F show a shared surface. The situation is additionally complicated when it is considered that not all 3D objects are box shaped – they may be surfaces, lines, complex 3D bodies and so forth.

A similar issue exists with respect to containment, as illustrated in Figure 5. Figure 5(i) shows a situation that is clearly identified as containment – the hatched box is completely surrounded by the clear box. However Figure 5(ii) shows a more ambiguous situation – the hatched box touches the side of the clear box.

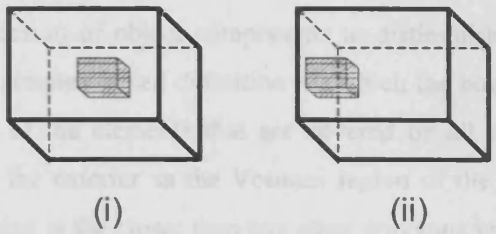


Figure 5 - Alternate Containment Relationships

2.3 Classifying Frameworks for Binary Topological Relationships

Topological frameworks provide a theoretical means to formally describe topological relationships between objects, giving a comprehensive method of differentiating between the various relationships such as those shown above. A formal understanding of geometrical relationships between spatial objects is a fundamental concept for the analysis of spatial data (Egenhofer and Herring, 1990). Frameworks also provide the groundwork for implementing topological modelling in a computer-based system, as in this case a series of well-understood rules is required to define and identify relationships between objects.

To distinguish between binary topological relationships, frameworks examine different characteristics of candidate objects for such relationships. A summary of these characteristics is presented here.

2.3.1 Whole-Object or Decomposition Based

The most basic differentiator between the various frameworks is described by Zhou *et al.* (2005) who note two categories for frameworks. The first group are based on the relationships between part components of objects –i.e. decomposition based, including those by Egenhofer 1989, Egenhofer and Herring, 1990, Clementini *et al.*, 1993, Chen *et al.*, 2001). In this case, each object (representing a real-world feature) is split into component parts – for example interior and boundary) and the relationship between each component part of object A and each component part of object B is examined. The results of these are then grouped to identify the relationship between the whole objects. The second group are those based on whole objects (including Cohn *et al.* 1997, Li *et al.* 2002) – in this case, instead of splitting the objects, the relationship between the objects as a whole is considered directly.

2.3.2 Variations on Component Definitions

Egenhofer and Herring (1990), Haarslev and Moller (1997) and Wei *et al.* 1998 intersect the interior, boundary and exterior of objects. A number of other options can also be identified when considering the selection of object components to distinguish topological relationships. Kainz *et al.* (1993) use a simplex-based definition for which the boundary is given in terms of the symmetric difference of the elements that are covered by all the triangles of the object. Chen *et al* (2001) define the exterior as the Voronoi region of the polygon – i.e. the area of influence where that polygon is the closer than any other polygons in the set. Pigot (1995) uses the boundary and co-boundary, the Set Theoretic operators defined by the OGC (see below) utilise the closure and the exterior. Billen *et al.* (2002) base their definitions on half-spaces.

2.3.3 Dimension of Intersection

Dimension extended models (which take into account not only the presence or absence of an intersection, but also the dimension of this intersection) include those described by Clementini *et al.* (1993), and Wei (1996). The Open Geospatial Consortium (OGC 2006) also takes this approach when defining their Full topological operators (see Section 2.4.1). This concept is also used as part of the Euler framework described by Zhou *et al.* (2005).

2.3.4 Number of Intersecting Components and Euler Number

As well as the dimension of intersecting parts, it is also possible to take into account the number of such intersecting components, as is described by Shi and Liu (2005), although this is

dependent on the shape of the objects themselves. Building on this, Zhou *et al.* (2005) describe an approach where the presence or absence of an intersection component is augmented by the dimension of the intersecting component and by the Euler number of that component. The calculated Euler number not only describes the number of connected components (as utilised by Shi and Liu 2005) but also describes the shape of the intersection – for example, are there any holes (the Euler number is described further in Appendix 1).

A number of frameworks for binary topological relationships are summarised in Table 1 below. Further details are given in Appendix 2.

Framework Name and Author	*Whole/Part Object*	*Dimension of Intersection Considered*	*Summary*
9-Intersection (Egenhofer and Herring 1990)	Part Object (interior, boundary, exterior)	No	Models relationships between the interior, boundary and exterior of each object. Concepts originate in point-set topology.
Dimensional Model (Billen *et al.* 2002)	Part Object – but parts are Dimensional Elements	Yes	Based on the concept of half-spaces, from which order and dimensional elements are defined. Models both the objects and the relationships using this approach.
Voronoi 9-Intersection (Chen *et al.* 2001)	Part Object (interior, boundary exterior)	No	Defines interior, boundary and exterior in terms of the Voronoi regions for each object.
Dimension Extended Model (Clementini *et al.* 1993, van Oosterom *et al.* 1994)	Part Object (interior, boundary, exterior)	Yes	Based on the 9-Intersection framework (Egenhofer and Herring 1990), but also takes into account the dimension of the intersecting components.
Boundary/Co-Boundary. (Pigot 1995 pg. 111)	Whole Object	No	Uses the union of two objects to create a third object. The boundary/co-boundary relationships are then used to differentiate relationships
SBox (Haarslev and Moller 1997)	Part Object (interior, boundary, exterior)	No	Utilises concepts of interior, boundary and exterior as presented in the 9-Intersection framework (Egenhofer and Herring 1990), but defines a generic subset of relationships to support qualitative spatial reasoning.
Region Connection Calculus (Cohn *et al.* 1997)	Whole Object	No	Designed to support qualitative spatial reasoning. Regions are considered the basic primitive for this framework. Can have any dimension but dimension of both regions in the relationship must be the same.
Poset/Order (Kainz *et al.* 1993)	Whole Object	No	Uses the concept of partially ordered sets (poset), and defines upper and lower bounds of these sets and a lattice (a poset in which every pair of elements has an upper and lower bound). These concepts in turn used to define topological relationships.

Framework Name and Author	*Whole/Part Object*	*Dimension of Intersection Considered*	*Summary*
Euler Number (Zhou *et al.* 2005)	Part Object	Yes	Extends the dimension-extended approach (Clementini *et al.* 1993), to consider the Euler number of each intersection component, along with the dimension of the intersection.
Object-Shape Framework (Shi and Liu 2005)	Part Object	Yes	Extends the 4-Intersection framework (Egenhofer 1989) to not only examine the presence or absence of intersection components, but the number and dimension of these components. Authors note that the number of intersection components between two circles is not identical to that between two ellipses, and that using these criteria, a total of 13 relationships between the ellipses can be identified, as opposed to 8 for the circles.
Part-Whole Framework (Price *et al.* (2001)	Whole Object for first level classification, part object for further classification	No	This framework considers both the intersection and difference between two objects, determining three relationships for objects A and B, namely A∩B, A-B and B-A. Combinations of these components are then used to define high-level relationships. The relationships further sub-divided into sub categories. For example, objects can be "connected with boundary-overlap" or "connected with interior overlap". Definition of interior and boundary is application-specific.

Table 1 - Topological Frameworks

2.4 Selecting a Framework

Three criteria were evaluated when selecting a framework to provide the foundation for this research, namely whether the framework complies with any existing standards for topological relationships, whether the framework handles 3D objects, and whether it handles the complex and compound objects encountered in many GIS applications. These are considered in turn.

2.4.1 Existing Standards

The ISO 19107 standard issued by the Open Geospatial Consortium (OGC 2006) in collaboration with the International Standards Organisation (ISO) is of relevance here. The Open Geospatial Consortium is a non-profit, international, voluntary consensus standards organization involved in the development of standards for spatial and location based services. It works with government, private industry, and academia to create open and extensible software application programming interfaces for GIS and other mainstream technologies.

The description of the standard includes a mechanism for characterizing topological relations as operators to be used in queries, although specific names for the operators are not defined. Three types of operators are suggested – Set Theoretic, Egenhofer and Full operators. The relations specified in ISO 19107 are only valid for point, curve, surface and solid objects as

> *"The theory for aggregate objects that are not homogeneous in dimension is not yet satisfactory enough to base a standard on."* (OGC 2006)

The Set Theoretic (or Boolean) operators utilise the *closure* (i.e. the union of interior and boundary) of the set. The standard then examines intersection between the closure and exterior of objects, forming a matrix of possible outcomes, as follows (where closure is denoted by Clo, exterior by Ext and ∩ represents the intersection operator).

$$R(A,B) = \begin{pmatrix} Clo(A) \cap Clo(B) & Clo(A) \cap Ext(B) \\ Ext(A) \cap Clo(B) & Ext(A) \cap Ext(B) \end{pmatrix}$$ **Set Theoretic Relationships**

The matrix can then be tested to determine if each intersection is empty or not, providing a total of 16 possible relationships or classes. If required, the Boolean template also allows the use of TRUE (intersection detected), FALSE (no intersection detected) and NULL (intersection not tested) options for each component, resulting in a total of 3^4 or 81 possible relationships (although where NULL is used relationships are not mutually exclusive). The use of NULL allows queries such as *find any objects where the boundary intersects the boundary of Object A* (whether the interior intersects the boundary or interior of A is irrelevant). This retrieves objects intersecting with A and objects adjacent to A and reduces the number of components of the matrix that require evaluation, thus improving performance.

The Egenhofer Operators test for relationships identified by the 9-Intersection framework defined by Egenhofer and Herring (1990), and function in a similar fashion to the Boolean Operators, with the major distinction that they distinguish between the interior and boundary of an object, thus resulting in the 3 by 3 matrix shown below, where:

$$R(A,B) = \begin{pmatrix} Int(A) \cap Int(B) & Int(A) \cap Bnd(B) & Int(A) \cap Ext(B) \\ Bnd(A) \cap Int(B) & Bnd(A) \cap Bnd(B) & Bnd(A) \cap Ext(B) \\ Ext(A) \cap Int(B) & Ext(A) \cap Bnd(B) & Ext(A) \cap Ext(B) \end{pmatrix}$$

9-Intersection Relationships

In the above Figure, Int(A) represents the interior of object A, Bnd(A) represents the boundary of A and Ext(A) represents the exterior of A. Again, an operator can be defined as above to test for particular spatial relationships between two objects. The symbol ∩ represents the intersection operator. For a basic implementation, each intersection can be TRUE or FALSE giving a total of 512 possible relationships, although many of these are not achievable in practice. However, the NULL (i.e. intersection not tested) option is also available. The inclusion of this option results in a total of 3^9 (19,683) potential relationships, again non-mutually exclusive.

The Full operators are implemented in a similar fashion to the Egenhofer operators, but the dimension of the intersection is also taken into account when considering the results. The intersection matrix is now a 9-character string with each character having the following potential values:

Symbol	*Non-Empty*	*Meaning*
0	TRUE	Intersection contains only points
1	TRUE	Intersection contains points and curves
2	TRUE	Intersection contains points, curves and surfaces
3	TRUE	Intersection contains points, curves, surfaces and solids
F	FALSE	Intersection is empty
N	NULL	Status of this intersection is not tested

Table 2 - OGC Clementini Intersection Pattern Matrix Values

The Full Topological Operators use an operator template which takes into account the dimension of the intersection – i.e. 0, 1, 2 or 3. False is returned if no intersection is detected, and NULL is returned if the intersection is not tested. This results in a total of 6^9 or 10,077,696 possible operator templates.

2.4.2 Handling 3D Objects

Of the frameworks identified in Table 1 above, many have been validated (through the systematic identification of all possible relationships described by the framework) only in 2D.

Chen *et al.* (2001) note that Voronoi 9-Intersection may be problematic to extend into 3D as the algorithms for 3D Voronoi diagrams are not as well developed. Further work is also required to validate the approaches taken by the Region Connection Calculus (Cohn *et al.* 1997), the Poset Framework (Kainz *et al.* 1993) and the SBox Framework (Haarslev and Moller 1997) a 3D context, although the relationships appear to be extensible to 3D.

A number of frameworks are, however, intrinsically dimension independent or have been developed specifically with 3D objects in mind, including the Dimensional Model (Billen *et al.* 2002) and the Part-Whole framework (Price *et al.* 2001). The Dimension-Extended model (Clementini *et al.* 1993) has been enhanced for 3D objects (van Oosterom *et al.* 1994 who also added an EQUALS relationship) as has the 9-Intersection framework Wei *et al.* (1998). For the latter, a test dataset (diagrammed by Zlatanova 2000, Chapter 6) is available to model a comprehensive set of topological relationships identified by this framework in 3D.

2.4.3 Handling Compound and Complex Objects

Three approaches can be identified for the determination of binary topological relationships between compound (multi-part) and complex (with cavities or holes) objects. Clementini *et al.* (1995) define a set of rules which utilise binary relationships at the component level to decide the relationship between the complex objects at higher level. Abdelmoty and El-Geresy (1995) represent a topological relationship between multi-part objects in the form of an intersection matrix, which details the relationships between all the pairs of object parts. Nguyen *et al.* (1997) attempt to combine these approaches in the context of the 9-Intersection Framework. They define the interior, boundary and closure for each object part (i.e. disconnected parts, each of which may or may not have holes) according to the following rules (A1 and A2 are two component parts of the same object):

$Interior\ (A1 \cup A2) = Interior\ (A1) \cup Interior\ (A2) \cup D(A1A2)$

where D(A1A2) represents:

- Any part of the boundary of A1 also forming part of the interior of A2.
- Any part of the boundary of A2 also forming part of the interior of A1.
- Any boundary parts shared between A1 and A2.

(i.e. all these cases are flagged as interior)

$Boundary\ (A1 \cup A2) = Bnd\ (A1) \cup Bnd\ (A2) - Int\ (A \cup B)$

$Closure\ (A1 \cup A2) = Closure\ (A1) \cup Closure\ (A2)$

Additional work has been carried out by Schneider and Behr (2006) who define a series of conditions for relationships between complex objects using the 9-Intersection framework, going on to enumerate the possible relationships between pairs of complex points, pairs of complex lines, pairs of complex regions and between object pairs of different types, using a proof by

drawing and constraint approach. Research described by Li (2006) forming part of a comparison between the relationships identified by the RCC8 (Cohn *et al.* 1997) framework and those identified by the 9-Intersection framework, gives a complete classifications of topological relations between multiple closed regions (which may be overlapping or disjoint), grouping each set of regions into a compound object, and considering whether the 9-Intersection components of the relationship between compound objects are empty or not.

2.5 The Selected Framework – 9-Intersection

Of the three frameworks identified by the OGC standards, the Egenhofer and Herring (1990) framework, known as the 9-Intersection framework, is "perhaps the most well-known topological formalism in geographic information science" (Li 2006). Additionally, this framework has been extended to support 3D objects, and work has also been done in relation to complex and compound objects, although to date this is only in the context of 2D relationships. Given this, the 9-Intersection Framework has been selected to underpin the determination of binary topological relationships as required by this research. The approach to handling complex and compound objects described by Clementini *et al.* (1995) has been selected for implementation as part of this research. In other words, if a primitive is flagged as a boundary component of a part object, then it is taken to be a boundary component of the object as a whole.

The framework has been further developed by the original authors (Egenhofer *et al.* 1993, Egenhofer and Franzosa, 1994, Mark and Egenhofer 1995), with a number of other authors basing their frameworks on the 9-Intersection (including Haarlem and Moller, 1997, Chen *et al.* 2001). A number of applications have also been developed around this framework (including Grigni *et al.*, 1995, Sun *et al.* 2002, Lin *et al.*, 2006, Papadias and Theodoridis 1997, Winter 1995, Kurata and Egenhofer 1996).

2.5.1 Limitations of the 9-Intersection Framework

Although the most appropriate framework to support this research, a number of limitations of the 9-Intersection framework can also be identified. The framework does not investigate the non-empty set results, either for their dimension or for whether they are disconnected or connected. Zlatanova (2000, pg. 131) notes that due to this issue there are at least eight different variations for some relationships in 3D that present identical results for the 9-Intersection matrix but are actually visually very different due to the configuration of their geometry. The framework does not handle relationships between more than two objects, and cannot distinguish between intersections having connected and disconnected elements.

Wei *et al.* (1998) have also noted that, for example, a closed line object does not have a boundary in topological terms. This implies that intersections between such object may return anomalous results – an object may intersect with the interior and the exterior of the closed line without intersecting with its boundary. Although this is theoretically correct, it may cause issues if a conditions-based approach is selected to minimise processing required for relationship identification. The exceptional nature of the situation means that it does not comply with many of the standard conditions (such as *A's boundary intersects with at least one part of B and vice versa* Zlatanova 2000, pg 113). See Section 7.3.3 for a description of the conditions-based approach.

Chen *et al.* (2001) identify an issue with the definition of exterior (which in the Egenhofer and Herring (1990) approach is taken to be the complement of the interior and the boundary) when the co-dimension of an object is not 0. When embedding a line in 2-dimensional space (i.e. co-dimension 1), the boundary (defined as the end Nodes of the line) does not separate the interior from the embedding space.

Finally, although investigation into relationships between complex and compound 2D objects has been carried out, this has yet to be extended into 3D.

2.6 Summary

This Chapter first reviewed existing topological frameworks, classifying them according to their use of whole/part objects, the inclusion of the dimension of the intersection and the consideration of the number of intersected parts.

The 9-Intersection framework was selected from those available as being most appropriate in the context of this research due to its inclusion in the ISO 19107 standard and to the work undertaken to evaluate the framework in the context of complex and compound objects and of 3D objects. A number of limitations of this framework, including potentially anomalous results where objects such as closed loops are considered, were also identified.

Chapter 3 combines the selected framework with identified end-user requirements, to define three generic queries for topology in GIS, proposing a mechanism to map the many framework relationships to end-user terminology.

3 Requirements Gathering

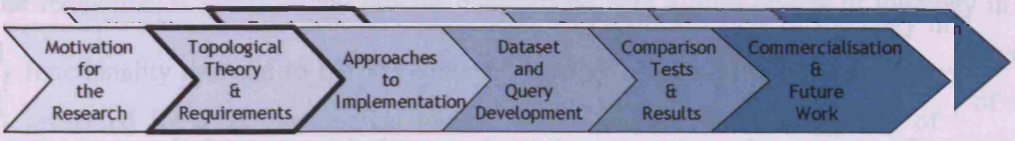


Figure 6 - Overview of Document Structure showing Context of this Chapter

3.1 Introduction

According to Gong *et al.* (2004) key problems in developing 3D GIS include 3D model design, visualisation and interaction. Billen and Zlatanova (2003) note that habitual use of 2D systems is also an inhibiting factor. To date, 3D GIS applications have commonly focussed on visualisation as evidenced in the literature (Dunbar 2003; Grunwald and Barak 2003; Nebiker 2003; McCann 2002; Janosch *et al.* 2001; Wust *et al.* 2004; Rowe *et al.* 2003; El-Hakim *et al.* 2003; Afshar *et al.* 2002). A further element can also be considered – a lack of understanding of the potential applications for 3D GIS, and in particular of the lack of appreciation of the analytical functionality that could be provided by 3D topology. The success of 3D GIS depends on its ability to perform the analysis that users currently expect from a 2D system.

The frameworks described in Chapter 2 provide a means to formally identify the many possible topological relationships between objects. However, they are generally derived theoretically from formal mathematical concepts, in absence of not taking into account end-user and application domain understanding of topological relationships. It is therefore important to express the relationships identified by the selected 9-Intersection Framework in these terms. A review of applications for 3D topology forms the starting point for this task, and is conducted in parallel with an investigation into current uses of 2D topology.

The results of the applications review process are combined to generate a set of functional requirements to be met by a 3D topological system. These are then grouped into those relating to the implementation of topology and data quality control, those relating to the identification of binary relationships and those relating to higher order relationships. The requirements relating to binary relationships (which form the foundation of higher relationships) are examined further, and a mapping from these requirements to a generic set of binary topological queries derived. Further details of the review process described here can also be found in Ellul and Haklay (2006).

3.2 Topological Functionality in 2D

The requirements identification process commenced with a brief review of topology in 2D to identify functionality relevant to the 3D context. Hoel *et al.* (2003) provide an overview of options presented by a 2D topological toolkit, which include the management of shared geometry, the definition and enforcement of data integrity rules, support for relationship queries, support for editing, the ability to reconstruct objects from unstructured geometry and the support for network analysis. Laurini (1998) also describes the use of topology to align and match spatial datasets held in different databases.

3.3 Analysis of 3D Applications

Research on 3D applications focussed on data quality issues and on applications where topological relationships such as adjacencies, connectivity and containment are utilised. Results derived from a review of a number of application sectors are summarised below.

3.3.1 Quality Control

The quality of the underlying data is fundamental to the relationship identification process and relates both to quality of the representation of single objects and of the relationships between them. To ensure that the results obtained are correct, object geometry must first be validated according to a pre-defined set of rules. Peters *et al.* (1994) list four topological characteristics of objects that can be measured and validated including "is the object manifold?", "what is the number of holes?", "are there any self-intersections?" and "are objects topologically identical?"

Euler equations (Appendix 1) can be employed to perform the first step of validation that an object is correctly structured - however Penninga (2006) notes that Euler Poincare holds for all simplicial complexes, including those with dangling Edges and Faces – i.e. this is a necessary but not sufficient test for object validity. Arens *et al.* (2005) add a test to validate that an object is closed to this list (this, along with a test for Face planarity, is described in Appendix 1). Van Oosterom *et al.* (1994) note that a polyhedron is valid if:

- An Edge is associated with only 2 Faces
- Faces do not cross or overlap
- All Faces are polygons

These tests make an assumption that a non-overlapping manifold object is the only valid object type – this may not always be the case, as is will be illustrated in Section 4.2 and Section 5.7.

Improved data quality required to correctly determine topological relationships has additional benefits in other areas of GIS, resulting in more accurate metric calculations (for example

distance between objects, or area of a land parcel) and greater map clarity (users are able to clearly identify the boundary of a land parcel).

When considering relationships between objects, it may also be possible to establish topology-based quality control rules. These could ensure that no two building polygons (or polyhedra in 3D) overlap, and define a process to correct the geometry should an overlap occur.

Lin *et al.* (1995) note that despite the general complexity of 3D geological models, there are some topological patterns that can be identified within a geological structure – for example, rock type A may always be found adjacent to rock type B. Similar quality control processes can be applied in terms of archaeological applications, with Jacobson and Vadnal (1999) describing the importance of ensuring that no unintentional gaps between objects are introduced into virtual reconstructions of archaeological sites. Although focusing primarily on visualisation of sites, they also outline the importance of object placement, i.e. the adjacency and containment properties of the objects in question. Stoter and Salzmann (2003) describe a number of situations relating to infrastructure above and below ground and discuss the conceptual requirement for a full 3D cadastre without overlaps or gaps in 3D space.

Within the context of object generalisation and multi-resolution display, topology can be utilised both to ensure that objects do not lose their intrinsic characteristics (holes and cavities) as coordinate information is removed by the generalisation process, and to ensure that adjacent objects maintain their relationship. An example of this is given by El-Sana and Varshney (1998), who describe the process of eliminating holes and cavities from 3D objects as they are generalised. They propose two approaches – preserving the genus (number of holes) of each object but simplifying the object using tolerance settings (the traditional approach) and the reduction of the genus of the object itself.

3.3.2 Analytical 3D GIS Applications

Earth Science applications, including geology, utilise 3D models extensively. Indeed, an early example of 3D geological modelling in a GIS context can be found in Carlson (1987), who uses a simplex-based approach to generate layers to support analysis of a geological model. Videla and Knox-Robinson (1997) describe functionality to identify both metric and topological relationships in the context of geological modelling. Apel (2001, 2006) lists a number of queries requiring identification of the topological relationships of adjacency and containment, including analytical queries such as "which model regions are cut by a particular fault?", "which Cambrian unconformities intersect Permian lime-stones?" and "select the set of geological objects with a given permeability and which have given faults as their boundaries". Similar

applications including key block analysis (Huseby *et al.*, 1997), engineering mining applications (Lixin and Wenzhong, 2003; Elkadi and Huisman, 2002; Elroi, 1998), oil and gas exploration (Belloso *et al.*, 1994) and Environmental Science (Sirakov *et al.* 2000) are also described in the literature.

Within the context of archaeology, Spikins *et al.* (2002) describe the importance of 3D analysis for the identification of phasing on pre-historical sites, again expressing a requirement for true 3D GIS tools. Barcelo *et al.* (2003) also describe the importance of identifying precise topological relationships when capturing 3D archaeological data, discussing processes involving analysis of artefacts found within and on top of a particular layer of stratigraphy. Gaiani *et al.* (2002) detail requirements for analysis as well as visualisation when using virtual worlds for archaeological restoration in Italy.

The cadastral and urban modelling fields provide additional literature in terms of applications requiring topological analysis of 3D objects. Modelling the increasingly complicated urban scene, including underground land use, multi-layer buildings such as those described in Grinstein (2003), and their corresponding usage and ownership cannot adequately be undertaken using 2D systems. Stoter and Salzmann (2003) discuss the conceptual requirement for a full 3D cadastre and provide a short list of suggested queries on the 3D cadastre, including finding neighbours, identifying which 3D objects are located on top of or under another one, and which 2D surface parcels intersect with 3D physical objects. Similar requirements are described by Onsrud (2003) and Koninger and Bartel (1998).

Support for Emergency Response teams is a specialist application within the urban modelling arena, with authors including Kwan and Lee (2005) and Takino (2000) describing applications requiring routing through 3D models of buildings for rapid determination of emergency exit paths. Both authors suggest a network-based analysis tool where a path through the 3D structure is modelled as a topological connectivity network – each room is represented as a Node on the network, and each path between rooms represented as an Edge between the appropriate Nodes. This approach allows the application of existing network and shortest path tools to the problem. A similar approach can be applied in the mining industry, where robots use a graph-based topological network to navigate through hazardous environments such as mine shafts (Silver *et al.* 2006).

3.3.3 Other 3D Analytical Applications

Gross (1998) and Kalra *et al.* (1995) describe the importance of topology with respect to the use of computer graphics in medicine, in particular to surgical simulation and modelling human

anatomy. Gross stresses that the topology and geometry of the model must be volumetric, as the simulator must model more than the surface. The simulator should also include functionality such as the repositioning of individual pieces of soft tissue. An application of 3D topology can also be found in Szymczak *et al.* (2006) who describe the use of a combination of Delaunay triangulation and graph-based algorithms to identify coronary vessel cores from a series of 3D Voxel-based images resulting from CT (computerised tomography) scans.

Brown (2002) describes the use of topology in identifying and distinguishing the chemical bonding properties of atoms, Martin (2000) describes uses of network topology in modelling protein structures and Kramer (2002) describes applications of topology to the modelling of biological development processes. Park *et al.* (2005) describe the use of binary topological frameworks to identify similarities and groups in 3D protein structures, examining the patterns or sequences of the binary relationships to compare the structures.

3.4 Classifying Requirements for Topology in 3D

Requirements identified by the above review process can be classified into three categories – Data Structuring and Validation, Binary Relationship Queries and Higher Order Relationship Queries.

Data Structuring and Validation relates to the application of topology to data quality control and the structuring of data into topological primitives. This includes the application of quality control rules such as '3D buildings must not overlap' during the process of structuring data into a format suitable for rapid querying of topological relationships. Binary Relationship Analysis, which relates to the analytical querying of the topological relationships between two objects (the main focus of this research). Directional queries have also been grouped as a subset of this type of query. These are described separately as they also involve non-topological properties of objects. Higher Order Functionality relates to applications and requirements examining the topological relationships between more than two objects or over entire datasets. Examples of this kind of functionality include network analysis and applications involving Voronoi diagrams and Delaunay Triangulations.

Table 3, Table 4 and Table 5 summarise requirements falling under each of these headings.

Topological Functionality	*Example Application Area*	*Reference*
Definition and Enforcement of Data Integrity Rules (includes management of shared geometry, support for editing). Support also required for integration with legacy 2D datasets.	General	Hoel *et al.* (2003), Galdi (2005) , Lin *et al.* (1995), Jacobson and Vadnal (1999), Brown (2002)
Describe the topological structure of an object – how many holes, tunnels, Faces etc. does it contain?	Object Generalisation and Multi-Resolution Geometry, Oil and Gas, Geology	El-Sana and Varshney (1998), Belloso *et al.* (1994)
Are there any gaps in this 3D cadastre?	Cadastral	Stoter and Salzmann (2003)
Ability to Reconstruct Objects from unstructured primitives.	General	Hoel *et al.* (2003)
Handle simple or complex geometry.	Cadastral, Archaeology, Hydrology, Oil and Gas	Sirakov *et al.* (2002), Spikins *et al.* (2002), Belloso *et al.* (1994), Koninger and Bartel (1998)
Requirement to model both curved and planar surfaces.	Geology	Apel (2006), Wust *et al.* (2004)
Handle relationships between 2D and 3D objects.	Cadastral	Grinstein (2003), Onsrud (2003), Koninger and Bartel (1998)
Custom Model Building – derive a Delaunay triangulation from the data, create a Voronoi tessellation of the data.	Biology, Geology	Kramer (2002)
Allow multiple models of topology to support different functionality. Allow pure topological model with no underpinning geometry.	Emergency Response, Chemistry	Kwan and Lee (2004), Takino (2000), Brown (2002)

Table 3 – Applications of Topology Related to Data Quality Control

Topological Functionality	*Example Application Area*	*Reference*
Identify adjacency of the different polyhedra in a model, and also between combinations of 0, 1, 2 and 3D objects and between complex objects.	Chemistry (atomic field modelling), Archaeology, Oil and Gas, Geology, Transport, Wireless packet routing, Environmental Science	Lin *et al.* (1995), Brown (2002), Belloso *et al.* (1994), Koninger and Bartel (1998), Nanda (2003)
Identify intersection between 3D, 2D, 1D, 0D combinations.	Cadastral, Geology, Transport, Environmental Science	Benhamu and Doytsher (1993), Huseby *et al.* (1997)
Identify containment of geometries of 3D, 2D, 1D and 0D objects and also of complex geometrical objects.	Archaeology, Oil and Gas, Cadastral, Geology	Barcelo *et al.* (2003), Jacobson and Vadnal 1999, Spikins *et al.* (2002), Belloso *et al.* (1994), Koninger and Bartel (1998), van der Molen (2003), Benhamu and Doytsher (1993)
Identify disconnectedness of two objects.	Geology, Oil and Gas	Apel (2001), Belloso *et al.* (1994)
Identify the specific topological relationship between two objects (3D, 2D, 1D, 0D combinations) and also of complex geometrical objects.	Archaeology, Cadastral, Geology, Hydrology	Barcelo *et al.* (2003)
Create rules to support binary relationship identification between complex objects.	Geology	Apel (2001)

Table 4 - Functionality related to Binary Topological Queries

Topological Functionality	*Example Application Area*	*Reference*
Identify and compare isomorphism of networks.	Chemistry (atomic field modelling)	Brown (2002)
Network Analysis, including finding shortest routes through building.	Wireless Packet Routing, Chemistry Geology Emergency Response, Mining	Hoel *et al.* (2003), Kwan and Lee (2004), Takino (2000), Silver *et al.* (2006), Brown (2002)
Matrix manipulation – create normalised incidence matrices for multiple connectivity and adjacency matrices.	Biology (embryo modelling)	Kramer (2002)
Matrix Manipulation tools - derive an adjacency matrix from a Delaunay triangulation and build a connectivity matrix from a Voronoi tessellation.	Biology (embryo modelling)	Kramer (2002)
Identify and compare sequences of binary topological relationships to determine protein similarity. Network of coronary vessel cores by using Delaunay triangulation and minimum spanning tree algorithms.	Biology (3D Protein Structures), Medicine (Coronary vessel reconstruction)	Park *et al.* (2005), Szymczak *et al.* (2006)

Table 5 – Functionality related to Higher Order Relationships

3.4.1 Directional Relationships

At first glance directional queries may also appear topological in nature - for example "is building A above garage B?" appears to be a directional adjacency query between these objects. In 2D, there is no widely accepted concept of direction, as "left" and "right" is entirely relative

to the direction in which a user is facing, although concepts of cardinal directions can be used and other mechanisms for defining direction between objects exist - Borrmann *et al.*, (2006) define concepts such as NorthOf and SouthOf). Abdelmoty and Williams (1994) associate four semi-infinite areas with each object, bounded by lines in direction North-East, North-West, South-East and South-West. Taking the simples approach, each object is reduced to a point through which these direction lines pass. A 4 by 4 matrix is then created to model the intersection of the direction lines of object 1 and those of object 2, as shown in Figure 7 and Table 6 (although polygonal, objects 1 and 2 have been reduced to a representative point, through which the directional lines are drawn).

Figure 7 – Two Objects with Direction Lines (adapted from Abdelmoty and Williams 1994)

	NE1	*NW1*	*SE1*	*SW1*
NE2	FALSE	FALSE	FALSE	FALSE
NW2	TRUE	FALSE	FALSE	FALSE
SE2	FALSE	FALSE	FALSE	FALSE
SW2	FALSE	FALSE	TRUE	FALSE

Table 6 - Direction Matrix for Figure 7

In 3D it is perhaps intuitive to describe a concept of above (further away from the centre of the Earth) and below. The review revealed that direction is an important component of a binary topological relationship, with queries such as "adjacent and above" and "adjacent and below" being common place in 3D (see for example Benhamu and Doytsher 1993, Barcelo *et al.* 2003, Stoter and Salzmann 2003, van der Molen 2003).

However, direction itself is not a topological concept. In the above example, Building A may be above Garage B but is not necessarily adjacent to Garage B, as there may be another object separating them. Defining direction is also complex - if a building is above another but offset to one side, is it still above? Li (2006b) includes directional relationships as a separate classification of spatial relationship types, along with topological relationships and distance measures, and gives examples of their usefulness in a 2D context – for example to deduce that "if San Marino is South of Germany, then Germany is North of San Marino". Note that for the latter example, the two countries do not have a border in common. Sharma (1996), Li (2006b), Mark and Egenhofer (1994) and Papadias and Theodoridis (1994) present results of work carried on directional concepts and combinations of directional and topological concepts in 2D.

3.5 *Grouping Requirements for Binary Relationship Identification*

Examining the requirements for binary relationships suggests that functionality related to the identification of binary topological relationships can be grouped into three query types. Firstly, a generic function to find intersecting objects can be identified. This relates to the identification of any objects having any non-disjoint binary topological relationship with a given object, and can be refined to identify objects only having a specific topological relationship with the given object (the second query type). Finally, functionality to identify the specific relationship between two given objects is also required.

The generic queries can thus be listed as:

- *Find Intersecting Objects.*
- *Find Objects having a specific Relationship with this one - abbreviated to Find Objects with Relationship.*
- *Find 9-Intersection Relationship between pairs of objects – abbreviated to 9-Intersection Pairs.*

The first two query types listed correspond directly to the "queries on whether two objects satisfy a set of topological relations" proposed by Clementini *et al.* (1994), with the generic *Find Intersecting Objects* providing a rapid mechanism to identify any non-disjoint objects without investigating the specific nature of the relationship. This type of query not-only provides support to the applications described above, but could also be important for data editing, to identify any objects that will be impacted should a specific object be edited. The third type of queries directly map to the second type proposed by Clementini *et al.* (1994) - "queries on the topological relations between two objects". Examples of domain-specific queries identified as part of the review process are mapped to these functionality groups in Table 7 below. Note that a number of these, particularly in the Cadastral application area, have been derived by extracting the topological component of a directional/topological query – thus

"Which buildings are adjacent to this garage?" has been derived from the end-user query "Which buildings are next to and on the side of this garage?".

Example Application Area	*Domain-Specific Query*	*Generic Query*
Geology	Which model regions are cut by a particular fault?	*Find Intersecting Objects*
Geology	Which Cambrian unconformities intersect Permian lime-stones?	*Find Intersecting Objects*
Oil and Gas	Which rock-types intersect with this drill path?	*Find Intersecting Objects*
Military	Which non-manifold objects are adjacent to this one?	*Find Objects with Relationship*
Archaeology	Which phase does this artefact belong to?	*Find Objects with Relationship*
Archaeology	What is the topological relationship between these artefacts?	*9-Intersection Pairs*
Cadastral	Who owns the buildings next to this one?	*Find Objects with Relationship*
Cadastral	Which buildings are adjacent to this garage?	*Find Objects with Relationship*
Electronics	Which processor chip is adjacent to this memory chip?	*Find Objects with Relationship*
Biology	What is the topological relationship between two components of a 3D protein structure?	*9-Intersection Pairs*
Archaeology	What type of garbage accumulation is adjacent to this one?	*Find Objects with Relationship*
Geology	What is the nature of the topological interaction between these two objects?	*9-Intersection Pairs*
Hydrology	What is the relationship between two units tagged 'permeable' and 'less permeable'?	*9-Intersection Pairs*

Table 7 – Grouping Functionality related to Binary Topological Queries

3.6 Mapping Requirements and Framework Relationships

"Topology as developed by mathematicians over the last century has shown the tendency [...] towards increasing abstraction and generality; much of it is thus of little or no obvious relevance to those attempting to formalise commonsense spatial reasoning" (Gotts *et al.* 1996). Although the limitations listed in Section 2.5.1 for the 9-Intersection framework would tend to lead to the development of a more complex framework, taking additional information such as the dimension of the intersection into account, the question should also be asked as to whether this would actually meet the needs of end-users of topology in GIS, and whether these users would actually require the GIS to distinguish between the resulting relationships.

Requirements for topology in are often not expressed directly in terms familiar to GIS developers. In fact, the topological significance of some analysis requirements above has been inferred from the descriptions provided. For example, descriptions using phrases such as "next

to", "underneath" and "beside" have been interpreted as referring to the topological concept of adjacency. In many cases, it can be said that users do not recognise the analysis they were describing as topological, due perhaps to a lack of familiarity with GIS in general and GIS-related topological terminology in particular. Authors of necessity focus on their own specific application area –phrases such as "constructions on top of each other" (Stoter and Salzmann, 2003) or "confirming the continuity of the C7 structure" (Belloso *et al.*, 1994) are used in the context of Urban Modelling and the Oil Industry respectively.

Two tightly interconnected issues can be identified in this regard. Firstly, the terminology used by developers and authors of frameworks to identify relationships differs from that employed by end-users, and also between end-users working in different domains. Secondly, many frameworks distinguish between a large number of binary topological relationships, whereas a smaller subset of these may be required by the end-user or relationships differentiated by the framework may have identical meaning the context of a particular application.

3.6.1 Differing Terminology

An example of this issue can be found in Shariff *et al.* (1998) who identify a series of different terms employed by end-users to describe a situation where a line bisects an area. These include *bisect, comes through, connected, connected to, crosses, cuts, cuts through, cuts across,* and *divides* amongst others. Similarly, the term *in* employed by Clementini *et al.* (1993) has the same meaning as *strictly contains* in Haarslev and Moller (1997).

A more general review of terminology encountered in papers relating to topological frameworks and their definition identifies phrases such as *inside, outside, intersects, is neighbour of* (Guting, 1988); *equals, disjoint, touch, within, overlap, cross, intersect and contains* Lin and Huang (2001); *strictly contains, strictly inside* (Haarslev and Moller 1997); *partially overlaps, externally connected* (Cohn *et al.* 1997). Similarly, Price *et al.* (2001) use phrases such as *connected with boundary overlap* and *connected with interior overlap* to describe relationships.

With specific reference to the 9-Intersection framework, Mark and Egenhofer (1995) describe work carried out using natural language terminology to represent the spatial relationship between various configurations of a road and a park area. As with the case above, a series of sentences were constructed, this time in both languages, to represent the relationships, and the similarity of responses obtained in English and Spanish was compared. Again, phrases such as *runs along, encloses, bypasses* and *circles around* were used. For 26 of the 107 sentences considered, more than 90% of users drew the same representation (topological relationship) – thus it would be safe to map the terminology in these sentences to the relationships identified.

However, a direct link between relationships and terminology was not as well established for other phrases. In fact, there were 20 sentences for which no single topological relationship emerged as a representation –it may not be possible to map every natural language sentence onto a single relationship (although a many:many mapping may be appropriate).

3.6.2 Number of Framework Relationships

The 9-Intersection framework identifies a total of 512 potential relationships, although the number of these realisable in practice depends both on the embedding space and on the nature of the objects (are they simple, complex, compound). However, many application domains group similar relationships. If, for example, the Figure below represents a person within a room, both relationships equate to *inside*.

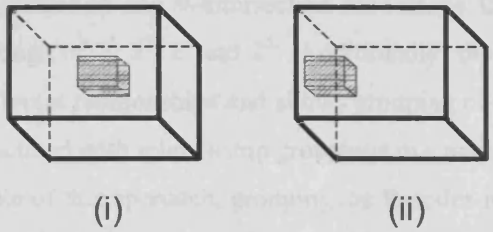


Figure 8 - Alternate "Inside" Relationships

3.6.3 A Proposed Approach

A mechanism is required to not only map framework relationships to end-user terminology but to allow this terminology to be defined on an application-specific basis. In addition, these relationships need to be evaluated in the context of the three binary topological queries identified as a result of the requirements review, namely *Find Intersecting Objects, Find Objects with Relationship* (find any objects having a specific relationship) and *9-Intersection Pairs* (find topological relationship between two objects). A number of authors have identified approaches to this - these generally involve classifying relationships into a small number of categories (Clementini *et al.* 1993, Sun *et al.* 2002, Shariff *et al.* 1998) or creating a hierarchy of relationships at different levels of detail (Billen *et al.* 2002).

Work undertaken by Kufoniyi (1995, cited in Zlatanova 2000) has led to the development of a simple, three-digit, numeric representation for each 9-Intersection relationship. This is known as a relationship code or R-Code, calculated according to the formula given in Table 8.

Table 8 – Example Decimal Encoding of 9-Interseciton Relationships (Zlatanova 2000)

Zlatanova (2000) lists advantages of the decimal approach, including the facilitation of comparison between 4-Intersection and 9-Intersection models, as the 4-Intersection model is represented by the weightings of 2^8, 2^7, 2^6 and 2^5. Additionally, the method provides a useful shortcut to identify the different relationships and allows grouping of these relationships. One or more R-Codes can be associated with relationship groupings in a many:one mapping. Zlatanova (2000) provides an example of this approach, grouping the R-codes into a total of 16 high-level headings including disjoint, contains, covered by and overlap. An extract from the results of her mapping process can be found in Table 9 below.

Table 9 - Mapping 9-Intersection Codes and Generic Relationships (from Zlatanova 2000)

This approach addresses framework complexity using terminology suggested by framework developers. It can be extended to a mechanism to allow users to define their own groupings, and to assign domain-specific terminology to these groupings. Table 10 shows the results of a sample definition process.

9-Intersection R-Code	*Formal Relationship*	*End-User Relationship*
R026 – R031	Disconnected	Not Touching
R179 – R223	Containment	Inside
R272 – R287	Adjacency	Touching but not Inside

Table 10 - Mapping 9-Intersection Codes and End-User Relationships

3.7 Reviewing the Requirements Analysis Process

As requirements analysis focussed on functionality that could be provided by using topology, requirements identified do not include those relating to data capture and data exploration (browsing) or other standard GIS tools that may be required in a 3D situation. Applications were examined at a conceptual level, focusing on the benefits that could be gained by users, without

considering availability of data or whether such applications could easily be implemented in practice. The review described above was not, however, confined to applications based on a particular approach to topological modelling, nor to traditional GIS application areas.

Another issue to consider is the most appropriate method to handle queries between legacy 2D data and 3D data as more and more of the latter emerges. This is particularly relevant with 3D building modelling, where sporadic buildings may exist rather than a continuous dataset. Two possible approaches exist, each of which requires further evaluation. The 3D data can be projected into 2D, and topological relationships then determined between two 2D datasets. Alternatively, the existing 2D data could be extruded into 3D and topological relationships identified in 3D.

In terms of mapping domain-specific terminology to framework relationships, the combined approach described above does not enforce the use of specific terminology, but allows expert end-users to map their own domain-specific phrases to individual topological relationships. Similarly, they can ignore any relationships identified by the framework but not relevant to their environment (by assigning a "not relevant" code). It does, however, assume that an expert user is available to undertake this mapping. An implementation of this approach is described in Chapter 7.

The review process is by no means complete. Many other potential application areas could benefit from 3D topological functionality in general and from the availability of 3D binary topological queries. It is likely that once basic tools have been implemented, it will be possible to approach potential end-users and to explore their requirements. The availability of basic demonstration software will help to ensure that respondents too have a good understanding of the nature of topological functionality and may thus be able to more-easily identify its relevance to their own application domain.

3.8 Summary

Following a review of existing 2D applications of topology in GIS, along with potential 3D applications, requirements for binary topological relationship querying in 3D were identified. In particular, three generic queries were listed. These are:

- *Find Intersecting Objects*
- *Find Objects with Relationships*
- *9-Intersection Pairs*

A mechanism to flexibly map application-specific and user-specific topological terminology to the many relationships identified by the 9-Intersection framework, using a unique numerical

value for each relationship was then given. The following Chapter considers existing approaches to the implementation of this functionality.

4 Existing Approaches to Implementation

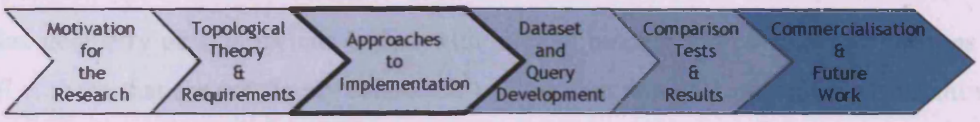


Figure 9 - Overview of Document Structure showing Context of this Chapter

4.1 Introduction

Chapter 2 presented a review of topological frameworks suitable for implementation in a 3D context, and Chapter 3 described a process to map the many relationships identified by the frameworks to a more user-focussed understanding of these relationships. This Chapter examines methods available for implementing the identified functionality focussing in particular on the 9-Intersection relationships (Egenhofer and Herring, 1990).

A brief description of some general characteristics of 3D objects which complicate the relationship identification process is first given. Two methods for relationship determination are then reviewed – As-Required calculation and the use of a structure-based approach. A review of the latter concludes that 3D Boundary-Representation (B-Rep) provides the most appropriate approach to underpin the work described in the remainder of this thesis, a review of existing B-Rep structures is presented.

A typical B-Rep structure (3D Formal Data Structure, 3DFDS, Molenaar 1992) is then examined in terms of its ability to model both the general characteristics of 3D objects and the specific 9-Intersection relationships identified by Zlatanova (2000). A revised version of 3DFDS, Extended 3DFDS, is presented to overcome some of the limitations of the Molenaar (1992) structure. The Chapter concludes by identifying two limitations of this revised structure in terms of rapid performance of 9-Intersection relationship determination queries – the number of relational joins to be followed, and the requirement for containment exception tables[2].

4.2 Characteristics of 3D Objects

A number of general characteristics of 3D objects must be taken into account by the relationship determination process. These are summarised here, and serve to illustrate the complexity of algorithms underpinning a 3D topological engine. Understanding the range of the objects to be

[2] Note that in an object-relational environment, foreign keys (and hence joins) can be replaced by references. Section 5.9.3 considers the implications of these in this context.

handled provides a means to evaluate any proposed approaches to implementation and validate that the approaches reviewed can handle objects found in a real-world 3D dataset.

Complex geometry objects include bodies with internal tunnels and cavities and surfaces with holes (i.e. those that are not simply connected[3]). This complexity is particularly relevant when validating the closure of a 3D shell to identify the interior, boundary and exterior of an object. The enclosed space inside a cavity or tunnel is defined as exterior to the object.

Curvature adds additional complexity to the process of representing geometry within and hence of identifying any areas of intersection. Representation methods include implicit functions (specifying the centre and radius of a sphere), polygon meshes (such as triangulations), parametric equations (specify a set of points and define a curve or surface to exactly fit these points), Bezier curves (specify start points, end points and guide points known as knots) or patches (complex curves or surfaces made up of many simpler curves or surfaces) including B-Splines. Non-Uniform Rational B-Splines (NURBS) are particularly useful when representing freeform surfaces, and are defined by a set of weighted control points (which determine the shape of the curve) and a knot vector (a sequence of parameter values which determine how the control points form the curve) and an order (which determines how many control points influence a given control point).

Overlapping objects should also be considered and can occur in datasets such as those describing urban structures. For example, multiple points may be co-located to represent a series of readings of pollution data. Objects such as furniture may share space with the interior of a room. Adjacent flats may share a common entrance, which is thus included in the Body representing each one. Concepts such as the overlay of a 3D Body representing a lake and a 2D Body representing a park of which the lake forms a part should be considered.

Compound objects are those made from combinations object parts often having different dimensions. These parts may be also disconnected from each other. Handling compound objects introduces additional code into the computational geometry process required for relationship determination as different parts an object may be related to other objects in different ways.

---

[3] An object is said to be connected if it cannot be built as the union of two non-empty disjoint open objects. This concept is applied in GIS to describe the structure of objects that are composed of single parts. Connected objects are simply connected if they contain no holes (Worboys 1995). More formally, an object is simply connected if it is path connected (i.e. any two points can be joined by a path) and every path can be continuously transformed into every other path (Brown, 1988).

Orientable manifold[4] surfaces are those where two sides can be distinguished (Hoffman 1989, pg. 38). This is important when considering the topological properties of closed 3D objects as an orientated surface allows the interior, boundary and exterior of the closed surface to be distinguished, facilitating the determination of the 9-Intersection relationship between such objects. Conversely, objects bounded by non-manifold surfaces complicate the identification of the topological relationships.

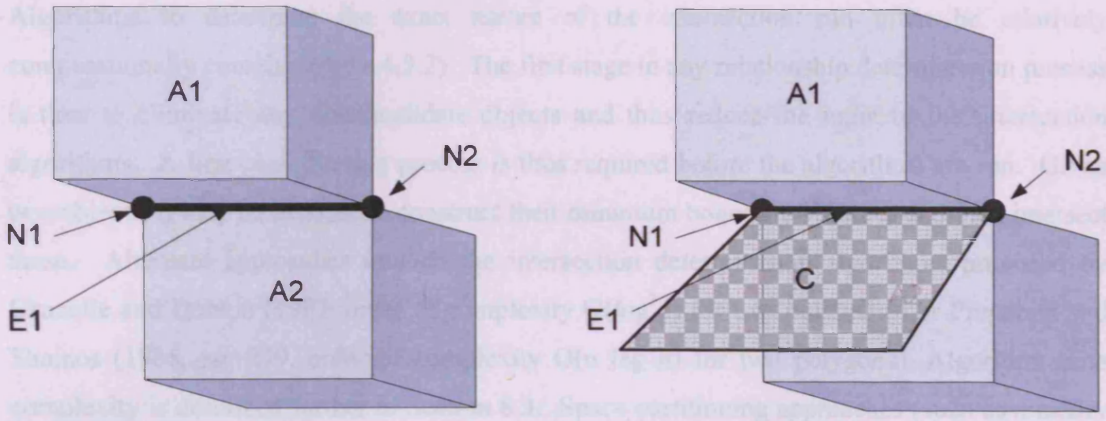


Figure 10 – Issue with Non-Manifold Objects

Figure 10 gives an example of the issue caused by non-manifold objects. In this case, whilst parts A1 and A2 of object A are manifold, the combined object A (which is a compound object) is not where A1 and A2 meet along Edge E1 the boundary changes from a 2D closed surface to a 1D line. Therefore whilst the relationship between A1 and C or A2 and C can be easily determined, as Edge E1 is a boundary primitive of A1 and A2, the relationship between the whole object A and C may differ depending on whether E1 is considered to be a boundary of A or not.

4.3 Identifying the 9-Intersection Relationships between Two Objects

A series of computational geometry[5] calculations are required to determine the exact nature of the intersection between two objects and hence identify the correct 9-Intersection relationship.

---

[4] Hoffman (1989, pg. 38) defines a manifold surface as having the property that around every one of its points, there exists a neighbourhood that is homeomorphic to the plane. That is, the surface can be deformed locally into a plane without tearing it or identifying separate points with each other. The boundary of a d-manifold with boundary is a (d-1) manifold without boundary. For example, a 3D cube, including its interior space, is a 3-manifold with boundary. Its boundary is given by a (3-1) 2-d manifold without boundary – i.e. a 2D surface with no Edges. Similarly, the boundary of a polygon (a 2-manifold with boundary) is a closed line (a 1-manifold without boundary). A ball (including its interior) is a 3-manifold with boundary. The boundary of this ball is a sphere –a 2-manifold without boundary.

[5] Computational geometry is defined by ISO 19107 (OGC 2006) as "the manipulation of and calculations with geometric representations for the implementation of geometric operations. Computational geometry operations include testing for geometric inclusion or intersection, the calculation of convex hulls or buffer zones, or the finding of shortest distances between geometric objects."

As these algorithms are implemented within a topological engine, they are beyond the scope of this thesis. However, a number of examples are presented here for illustrative purposes. The focus here is on the determination of the 9-Intersection relationship between two 3D objects. This involves identifying the interior, boundary and exterior of these objects.

4.3.1 Eliminating Non-Intersecting Objects

Algorithms to determine the exact nature of the intersection can often be relatively computationally complex (see a 4.3.2). The first stage in any relationship determination process is thus to eliminate any non-candidate objects and thus reduce the input to the intersection algorithms. A first-pass filtering process is thus required before the algorithms are run. Given two objects, it may be feasible to construct their minimum bounding volumes and then intersect these. Alternate approaches include the intersection determination algorithms proposed by Chazelle and Dobkin (1987, order of complexity $O(\log^3 n)$ for two polyhedra) or Preparata and Shamos (1985, pg. 279, order of complexity O(n log n) for two polygons). Algorithm time complexity is described further in Section 8.3. Space partitioning approaches (such as a binary space partitioning (BSP) tree, which is a recursive hierarchical subdivision of space into smaller convex sub-spaces, or the creation of an Octree (Hoffman 1989, pg. 62) which is based on similar principles but sub-divides the space recursively into eight equally-sized cubes) may also be applied. A separating plane test may also be used to determine disconnectedness does a plane exist that has the vertices of one object on one side and those of the other on the other side?

However, given the object-relational database to be used as an implementation environment (described in Appendix 8), the filter is likely to take the form of an R-Tree index query to eliminate any non-candidate objects (see Section 4.4.1 for a description of the R-Tree index).

4.3.2 Determining the Intersection – 2D

Konidaris *et al.* (2003) propose an algorithm for Edge segment adjacencies (to identify adjacent polygons) which involves splitting polygons into lower and upper chains, identifying the middle edges of each chain, computing the mid-points of the middle edges and then determining the angles formed between the line segment joining the two mid points and their middle edges. If the two resulting angles are both less than 180° then the intersection of the middle edges is computed and the edges on the opposite side to the intersection point discarded. If the middle edges are parallel, then adjacency is identified if they form part of the same line. If at least of the angles is greater 180°, then the line joining the mid points passes through at least one of the polygons, and the edges on the opposite side of this line are discarded. This process is repeated iteratively until one of the chains has fewer than four edges remaining, when standard edge

intersection is applied – i.e. the main purpose of the algorithm is to reduce the number of full edge intersection computations that are required. Algorithm complexity is given by the authors as O(log n) where n is the upper bound of the number of edges in each chain.

4.3.3 Determining the Intersection – 3D

Taking a crude approach in 3D (Preparata and Shamos, 1985, pg. 307, also described by Nguyen *et al.* 2005), determining the 9-Intersection relationship between two bodies involves calculating the signed distance between each Vertex bounding the planar polygons forming Body A and the planar polygons forming the Faces of Body, using repeated iterations of the vertex-to-polygon algorithm shown below[6] (Figure 11, Schneider and Eberly, 2003, pg. 386). A number of outcomes of this process can be identified. If all distances between the Vertices and Faces are non-zero and positive and negative distances identified for different Vertices of the same Face, then all components of the 9-Intersection relationship are NOT NULL and this represents an Overlap relationship (R511). If all distances are 0, then this is an Equals relationship (R200). This algorithm has complexity $O((n)^2)$, where n is the total number of vertices for both polyhedra.

Muller and Preparata (1978, cited in Preparata and Shamos 1985, pg 307) propose an alternative approach in 3D, as follows (the algorithm description has been adapted from Hasegawa and Sato, 2004 and Preparata and Shamos, 1985):

- Find a point p that is shared between the two polyhedra (P and Q), by projecting the polyhedra onto a 2D plane and finding the intersection of the resulting polygons. This common point can be considered as the origin and can be found by (Preparata and Shamos 1985, pg 309):
    - Projecting the intersection of polyhedra P and Q with their vertical planes of support onto the horizontal plane.
    - The intersection of the resulting convex polygons is then determined, and vertical segments projected upwards from all points inside this intersection to re-intersect with P and Q.
    - The required point p can be found when two specific conditions are met – a point *a* in the horizontal plane is identified where the segment passing through P does not intersect that passing through Q, and a point *b* where the two segments do overlap.
    - Point p can be found by joining the top of the segment through P at point *a* with the bottom of the segment through P at point *b* and performing the same operation for the segments through Q. Point p is the intersection of these two lines.
- Using this origin p, convert the planes of each convex polyhedron into a series of vertices by dual transformation. A plane of ax + by+ cz = 1 (where this plane equation

is defined relative to the origin identified above) becomes a point (a,b,c). Using this transformation, planes at a distance *l* from the origin become points at a distance 1/*l* from the origin.

- Find the minimum convex polyhedron that includes all these vertices (using a convex hull algorithm, Preparata and Shamos, 1985, pg 95).
- Convert the convex polyhedron by dual transformation back to the geometry of the common (shared) part.

This algorithm yields the Vertices and Faces of the contact volume. The implementation makes an assumption that each polyhedron is represented using the Doubly-Connected Edge List (DCEL[7]). The algorithm has complexity O(n log n), where n is the total number of vertices in both polyhedra (Preparata and Shamos, pg. 315).

Other algorithms for the intersection of 3D polyhedra include a Space Sweep algorithm (Hertel *et al.* 1984) with a complexity of O (n log n). This extends the 2D concept of plane sweep (where a line is moved from left to right across the figure) into 3D. In 3D, this line is converted to a plane. Dobkin and Kirkpatrick (1983, cited in Hertel *et al.* 1984) also present an algorithm with complexity $O((\log n)^2)$ after $O(n^2)$ pre-processing. Table 11 summarises the algorithm complexity for each algorithm. Implementations of the relationship identification process would use the least complex approach. Commercial implementations may also have developed more efficient proprietary algorithms.

Algorithm	*Complexity*	*Comment*
Nguyen *et al.* 2005	$O(n^2)$	
Muller and Preparata 1978	O(n log n)	
Hertel *et al.* 1984	O(n log n)	
Dobkin and Kirkpatrick 1983	$O((\log n)^2)$	$O(n^2)$ pre-processing

Table 11 - Algorithm Complexity for 3D Intersection Determination

Handling real world 3D data requires that the algorithms above take into account the characteristics of 3D objects. Bodies may be formed of large numbers of coordinate tuples and polygons –each tuple must be tested with each other polygon. Algorithms must be extended to consider situations where the captured data contains errors. If the coordinates captured for A do not exactly match those for adjacent object B (perhaps due to capture techniques or rounding errors), then some tolerance levels must be set within which a match can be made. Other structures, particularly in geological or petrochemical datasets, are not necessarily rectilinear or box-shaped. Algorithms must take into account the characteristics of 3D objects listed above.

[7] DCEL is an edge-based data structure where each edge is in turn formed by two directed half-edges. Each half edge has an origin and destination, and the structure stores three sets of records – vertices (storing the coordinates), edges (pointing to the opposite-edge, the left and right faces and the previous and next edges) and faces (pointing to the single edge for which they are the incident face).

Implementing these algorithms in a computerised environment may also cause issues due to the finite nature of the digital environment. Errors (described in further detail in Schneider and Eberly (2003, pg. 3) include issues such as order of addition, rounding errors, cancellation of significant digits during subtraction of nearly equal numbers and division by numbers close to zero. Additional issues may be caused by the geometrical representation within the database of the objects themselves – two objects may appear adjacent on screen, but their underlying coordinates may differ slightly, due to the data capture process. Thus incorrect results may be returned by the relationship identification algorithms. Thompson *et al.* (2006) note that such issues may invalidate the analysis, and additionally note that if a surface is described by more than 3 points, as it may then not be planar. Thompson (2005) proposes an alternative framework for implementation, based on regular polytopes (see Appendix 2 for a more detailed description of this approach) and extend this to an approximated polytope (Thompson *et al.* 2006), which makes use of an approximated point rather than the rational points generally used. The use of integers to represent the approximated points further enhances this approach.

Computing the Signed Distance from a Vertex to a Polygon

1. Identify the outward pointing normal vector to the polygon plane $\vec{n}$. This can be done by taking the cross product of vertices as follows:

$$\vec{n} = (V1 - V3) \times (V2 - V3)$$ **Cross Product to define a Normal Vector**

2. Project the vertex P onto the plane in which the polygon lies, at P'
3. Project each vertex (V1 to V5) of the 3D polygon and P' onto a 2D plane such as the XY plane
4. Compute the point Q'' on the projected polygon closest to the projected point P''.
5. Reverse the projection process, and identify Q', the point on the 3D polygon closest to P'. Note that if P' falls inside the polygon, the Q' = P'.
6. Determine the signed distance between the original point P and Q'. This is the distance between the point and the polygon. Any points on the side of the polygon plane indicated by the direction of the normal $\vec{n}$ are assigned a positive distance from the plane. Points on the underside of the plane will be assigned a negative distance.

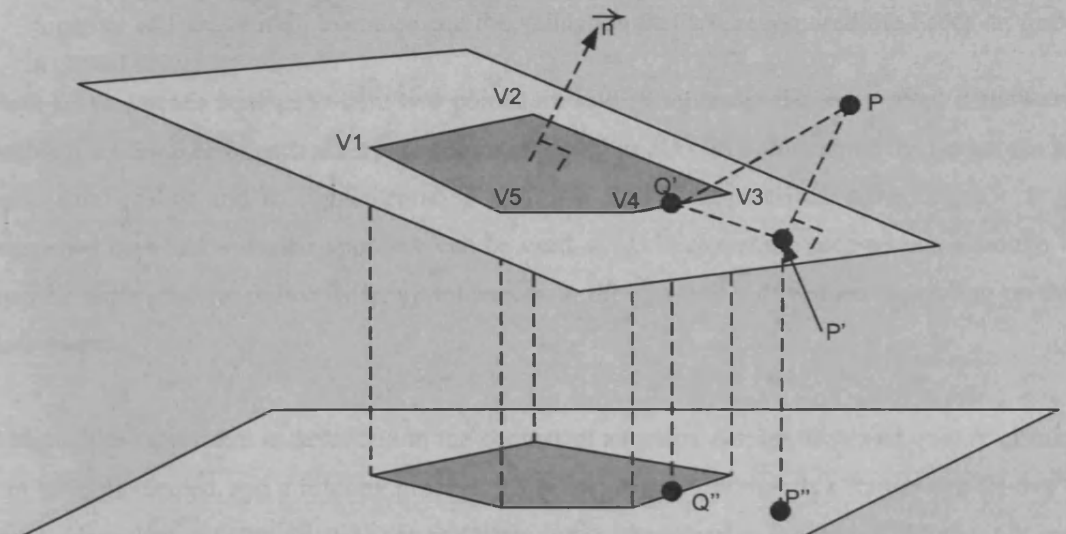


Figure 11 - Determining the distance between a point and a polygon

4.4 Approaches to Relationship Determination

Two approaches to the implementation of such computational geometry algorithms can be identified – As-Required calculation and the use of topological data structures. In the former, relationships are determined only when requested by the user, utilising the 3D coordinate information directly to implement the calculation. In the structure-based approach, relationships are pre-determined as the data is captured and the results stored in a topological data structure, which can then be interrogated. Should the underlying data change, the topological data structure is updated accordingly. An overview of both approaches is presented here.

4.4.1 As-Required Implementation

Given their proprietary nature, algorithms for As-Required topological relationship calculation are difficult to locate. However, Krivograd and Zalik (2000) describe a four-step approach to the determination of topological relationships and population of a topological data structure in a 2D context. The first two of these steps relate to data quality improvement, and can be applied to the As-Required calculation and extended to 3D. These modified steps are given as follows:

- Step 1 – Detect and remove any inconsistencies. In 2D, this relates to the splitting of all input lines into line segments. In 3D, this operation is extended to the splitting of input polygons into single Faces.
- Step 2 – Improve the quality of the data, which involves making sure that line segments within a certain threshold of each other are given as intersecting, that lines that are within a certain distance are merged, that isolated points are removed if required, that points within tolerance are snapped together and so forth. For 3D, the quality improvement process must also be applied to the identification of common line segments between Faces, the snapping together of Faces within tolerance and the validation that where required the Faces do form a closed object.

Note that tolerance settings (where two points are said to represent the same point if they are within a set distance of each other) are commonly used in 2D GIS to overcome the issues due to poor data quality and to implementation within a finite computerised environment. It is suggested here that a similar approach can be used in 3D to overcome such issues, although it may be appropriate to define different tolerances in the x, y and z directions depending on the data source.

Although the algorithm is described in the context of an entire dataset, localised quality checks can be implemented, and a filtering process, using spatial indexes such as a Range-tree (R-tree[8]) index (described in Guttman 1984) or an Octree index (described in Hoffman 1989, pg. 62, and in Samet 1995) could be used to eliminate data not relevant to the specific query. In general, clustering algorithms are used to match Nodes in the dataset and eliminate redundant points

---

[8] R-trees in 3D consist of overlapping boxes that represent geometrical objects or groups of such objects. To search the tree for a particular object, all the children of the root whose directory box contains the object are first visited. All sub-trees of these children are then visited, and the process repeated at each level of the tree until the leaves are reached. Any objects on the selected leaf Nodes are then candidates for intersection with the object in question (Rigaux *et al.* 2002).

Kofler *et al.* (2000) note that R-trees allow the data to be indexed very naturally. They propose two advantages of this index type – they are very flexible, and can incorporate multiple data types, and multiple levels can be introduced to represent a hierarchy of situations that may not correspond to the regular areas required by the Octree. Additionally, an R-tree is a depth-balanced tree (i.e. the number of levels of child-Nodes is consistent throughout the tree) ensuring equal performance for searches throughout the tree. Index records in its leaf Nodes contain pointers to data objects (Guttman 1984) and most of the Nodes in an R-tree are at leaf level. Thus it may be appropriate to hold all the non-leaf Nodes in memory to improve index performance (Blackwell 1987).

(Chrisman *et al.* 1992). Ware and Jones (1998) extend this process to take into context errors beyond those at vertices, considering Edges within the context of the object as a whole.

Once the data has been cleaned, algorithms such as those described in Section 4.3.3 are then implemented to determine the 9-Intersection relationship.

4.4.2 Data Structure Implementation

The data structure approach calculates relationships once and queries the result many times. A topological engine makes use of computational geometry algorithms to determine the binary relationship, the results of this calculation stored in the data structure. It is also possible to use these structures to model topological relationships were underlying geometry does not exist.

A number of approaches to the design of the structure exist. The creation of a simple table of relationships listing the exact topological relationship (9-Intersection R-Value) between each pair of objects is possible. Similarly, a graph approach (where each adjacency relationship is stored as an Edge and each object is a Node) can be considered. For field based datasets, topological relationships can be derived through the examination of specific graph types including Voronoi diagrams and Delaunay Triangulations (Worboys and Duckham 2004). Matrix representation (where the objects form the row and column headings of the matrix and values are set to NOT NULL if adjacency is true) is also available. Boundary-Representation (B-Rep) can also be considered. Hoffman (1989, pg. 37) presents properties of B-Rep model including:

- It can represent a solid unambiguously by describing its surface and topologically orientating it such that at each surface point the side on which side the solid lies can be identified.
- Two parts exist in the model – the topological description of connection and orientation of Edges, Nodes and Faces and a geometric description to embed these elements in space.
- The Topological Description consists of Nodes, Edges and Faces specified abstractly and their incidences and adjacencies indicated. Edges are described by Nodes, and Faces by an ordered list of Edges.
- The Geometrical Description can consist of plane equations of the surfaces of which the Faces are a subset, along with the coordinates of the Nodes. In the equations, at point p in the interior of Face f the surface normal points to the exterior of the solid.

Although not described in Hoffman (1989), in a GIS context, the geometrical description of the primitives can also be provided through the use of spatial object types available within Object-Relational databases.

4.4.3 Selecting a Data Structure

Table 12 compares the list, graph, matrix and B-Rep approaches. It can be seen that storing a list or table of relationship R-Values (having columns Object ID 1, Object ID 2, R-Value) provides the most efficient approach for the representation of topological relationships between 3D data within an Object-Relational database. This provides both minimal storage required and rapid query performance. However, when considered in a GIS context, the B-Rep approach offers a number of advantages that outweigh its additional data storage and data maintenance requirements:

- Data quality control is considered an important aspect of the implementation of topology within a GIS context. Although it is possible to separately implement such processes in conjunction with a list, graph or matrix approach (or for the As-Required approach) data quality control is directly implemented as part of the topological engine population process for a B-Rep structure.
- Using the list-based, matrix or graph approaches the geometry of objects cannot be reconstructed from the topological primitives.
- In the context of compact (bounded and closed) manifold objects, Tse and Gold (2003) note that advantages of the B-Rep model include the fact that operations on B-Rep structures are guaranteed to preserve mesh connectivity as Euler-Poincare connectivity (Appendix 1) is maintained.
- The existence of multi-part (compound) objects was also identified as a characteristic of 3D objects. In the case of the list, graph and matrix approach, there is no simple mechanism to determine and store relationships between individual part objects as well as between the parent objects. Although it may be possible to model each part object as a Node on a graph, these parts must then be linked to the parent object and this parent/child relationship differentiated from the topological relationship modelled by the Edges of the graph.
- B-Rep offers the ability to store relationships identified by other frameworks without any additional computational geometry requirement. This includes those identified by the ISO 19107 frameworks described in Chapter 2 (i.e. Set Theoretic and Full).

Additionally, B-Rep structures can be represented easily in Object-Relational databases and can take advantage of the spatial object types available in such environments.

Selection Criterion	*List of Relationships*	*Graph*	*Matrix*	*B-Rep*
Object-Relational Representation	Lists can be directly represented as tables.	Binary relationships can be stored within a simple table. [9]	Matrices not generally represented in a database using standard n by n tables due to low density of data. It is also difficult to extend the matrix as new objects added. Sparse data structures may be used to overcome some of these issues. – in this case, only non-disjoint relationships are stored.	Primitives can be directly represented as tables.
9-Intersection Relationships	Relationships can be stored as attributes of each object pair.	Can be stored as attributes on the graph Edges. If no Edge exists, relationship is disjoint.	Can be stored as an entry in the matrix, redundant storage where relationship is disjoint.	Can be derived through Structure Queries.
Relationships between sub Parts of Compound Objects	Not Directly Available. May need multiple lists.	Not Directly Available. May need multiple graphs.	Not Directly Available. May need multiple matrices.	Relationships between two part objects, or between a part and a whole object, can be queried from the data structure.
Population Time for Model	High. 9-Intersection relationship must be determined at the outset.	High. 9-Intersection relationship must be determined at the outset.	High. 9-Intersection relationship must be determined at the outset.	Medium. Population algorithm is only required to identify shared primitives. 9-Intersection relationship is determined at query time.
Data Quality Control implicit in relationship determination algorithm	No, but can be included explicitly.	No, but can be included explicitly.	No, but can be included explicitly.	Yes – quality issues such as overlaps identified as part of the process of populating structure with shared primitives. Primitives can be visualised to support any data correction required.
Storage	Generally low. However, size of lists depends on number of	Depends on whether a simple graph (low, but see discussion	High, (n * n) where n is the total number of objects, assuming	High (however linearly proportional to the number of

[9] Note that as higher-order relationships are not included as part of this research, recursion does not present an issue here.

Selection Criterion	*List of Relationships*	*Graph*	*Matrix*	*B-Rep*
	interrelated objects. If one object intersects with 20 others, there will be 20 entries in the list. Multiple lists may be required, one for each selected Framework.	for lists) or triangulation (high) approach is selected. Multiple graphs may be required to model different frameworks.	that disjoint relationships explicitly stored. However, data structures taking advantage of the sparse matrix (i.e. where most entries are 0, or in this case represent disjoint relationships) may reduce this to n. Multiple matrices are required for different frameworks.	objects), particularly if primitives are also stored as spatial objects. One structure can support many different framework queries.
Ability to reconstruct objects from the topological primitives.	None.	None.	None.	Objects can be reconstructed.
Ability to model other topological frameworks.	New list required.	New graph required.	New matrix required.	Existing primitives can be reused for frameworks referencing interior, boundary, exterior or closure of objects. Frameworks requiring dimension and number of shared primitives are also supported.
Query Performance Time	Fast – single SQL query of simple table.	Fast – single SQL query of simple table.	N/A (Matrix cannot easily be modelled within a relational database environment).	Medium – multiple tables must be queried. Procedural language such as PL/SQL also required.
Handling Curved Surfaces	Yes, depending on relationship determination algorithm.	Yes, depending on relationship determination algorithm.	Yes, depending on relationship determination algorithm.	Yes, provided that the surfaces can be described in terms of Node, Edge and Face primitives.

Table 12 - Selecting a Data Model

4.5 *Comparing the As-Required and B-Rep Approaches*

Validating data quality is necessary for both B-Rep and the As-Required structure, in order to ensure that query results are correct for topological and metric queries. Thus Steps 1 and 2 of the process described in Section 4.4.1 must be executed for both data structures, either on data capture or before relationship determination. B-Rep structures offer advantages in terms of data quality control, as this process forms an intrinsic part of the structure population and maintenance process. Once validation has taken place, it is a relatively simple operation to populate the B-Rep structure with the resulting primitives, resulting in a relatively low structure population and maintenance overhead.

Given good data quality, for the As-Required approach, the computationally-intensive coordinate geometry algorithms (such as that shown in Figure 11, taken from Schneider and Eberly, 2003, pg. 386) required for relationship determination are run many times, potentially slowing down system performance and utilising system resources. This is particularly the case where multiple users are performing analysis on the same database and the algorithms are implemented to run on the database server. In the case of B-Rep, however, computationally intensive coordinate geometry algorithms are not required to determine 9-Intersection relationships – instead, queries against a non-spatial relational data structure are executed (note that extensions are required to the standard B-Rep structure to identify all relationships – these are described in Section 4.7).

The use of appropriate spatial indexes may, however, significantly improve performance for the As-Required calculation, to the point that these could be more appropriate when data is frequently updated or modified. Thus As-Required calculation has advantages in situations where data is undergoing continuous modification – in this case, it is more efficient to calculate relationships in memory rather than repopulate the topological data structure each time the geometry changes.

Elimination of redundant storage is often mentioned in the context of B-Rep structures. This is countered, however, by the time required to reconstruct the whole object from the Node primitives, for visualisation and metric calculation purposes. It may therefore be more appropriate to use the topological structure for topological relationship determination and data quality and a non-topological structure to assist rapid visualisation of data. However, advantages of topological structures in terms of storage are lost using this approach.

Table 13 summarises the comparison criteria described above.

Step	*B-Rep*	*As-Required*	*Performance Comparison*
Data Quality Control (either as a batch process or on data capture)	Steps 1 and 2 from Section 4.4.1	Steps 1 and 2 from Section 4.4.1	Identical. Data quality control required for both approaches to ensure that metric and topological query results are correct.
Structure Population	Use quality control primitives. Populate the data structure and update the database with the corrected data.	Update the database with the corrected data.	Process likely to be slower for B-Rep approach. Additional tables must be populated and spatial and non-spatial indexes updated.
Binary Relationship Query	Relationships can be queried directly from the structure.	Coordinate geometry algorithms required (see Section 4.3.3)	Process likely to be slower for As-Required, particularly given additional complexity of 3D data.

Table 13 – Comparing B-Rep and As-Required

A consensus has not yet been reached on the optimal approach even in 2D context, with vendors offering varying implementations (ESRI 2006 offer an As-Required implementation, Laser-Scan 2007 and GE Network Solutions 2004 utilise a data structure). Baars *et al.* (2004) compare the ESRI and Laser-Scan approaches for functionality, noting that the ESRI approach offers users the possibility to have greater control over rule definition and offers a short validation process. With Laser-Scan rules are centrally defined and enforced, and the software is not GIS specific. Van Smaalen (2003) notes that data models with stored topology "appear to become scarcer amongst recently developed GIS data models" and ISO 19107 (OGC 2006) states that relations can be calculated either by using set theoretic operations defined on the on the geometry or algebraic operations defined on the topological structure.

Taking the 3D context into account, Zlatanova *et al.* (2004) conclude that a 3D topological structure will perform more efficiently for neighbourhood queries such as navigation and provides benefits in relation to maintaining data consistency. The additional complexity of 3D computational geometry algorithms adds weight to this argument. However, comparative statistics have not yet been determined for 3D data, as implementations of an As-Required system in 3D are, as yet, unavailable. Chapter 8 describes tests to compare a Proxy for As-Required queries and B-Rep approach, providing further insight into this debate.

4.6 A Review of Existing B-Rep Structures

A summary of characteristics of B-Rep data structures is presented in Table 14 below.

Characteristic	*Options*	*Relevance*
Simplex Primitive Type	Simplex or Not	Number of primitives and required storage increases for simplex-based structures, and additional algorithms may be required to restructure the data into simplexes (including TINs and tetrahedra). However, no algorithms required to validate planarity of the individual Face components. May be more suitable for continuous datasets (field data).
Containment Exceptions	True or False	Can the structure handle violations to the NODE/EDGE/FACE hierarchy? This object is required to model a point object such as a light fitting located directly on a Face such as a wall. Note that these are not required for a simplex-based approach, as the Face would be triangulated in such a way to as accommodate the light fitting.
Compound Geometry	True or False	This is required where an object is made up of a mixture of geometry types (points, lines, surfaces and bodies) or where an object has multiple disconnected parts.
Edge Primitive Defined	True or False	This impacts storage requirements for the data structure, but also impacts query performance – if an Edge object is not defined, then it must be artificially generated from the end Nodes – this may cause performance issues when 9-Intersection queries are run on the interior of Line objects.
Tunnels, Cavities and Holes	True or False	Can the structure model internal cavities and holes or tunnels through the 3D object? This is important to represent real-world objects, which tend to be complex in structure.
Curved Surfaces	True or False	Can the structure handle curved surfaces – particularly relevant for Earth-Science applications? This identifies whether the structure was developed to support regular structures with planar Faces such as urban modelling or irregular curved structures such as geology. These are currently modelled by Triangulated Irregular Networks (TINs) in B-Rep as B-Rep does not support curved Faces.

Table 14 - Summary of Characteristics of 3D Topological Data Structure

Table 15 summarises the implementation of these characteristics by a number of B-Rep structures reviewed as part of this research. A brief description of each of these structures is given in Appendix 3, which also lists 3D graph-based structures and object-oriented structures.

Data Structure Name	*Author (s)*	*Simplex Primitive Type?*	*Containment Exceptions?*	*Compound Geometry?*	*Edge Primitive Defines?*	*Tunnels, Cavities and Holes?*	*Curved Surfaces?*
3DFDS	Molenaar (1990)	False	True	False	True	True	Planar
(Unnamed)	Tse and Gold (2003)	True	False	False	True	False	Both
TEN	Pilouk (1996)	True	False	False	True	True	Non-Planar
SSM	Zlatanova (2000)	False	True	False	False	False	Planar
VPF+	Ladner *et al.* (2001)	False	True	False	True	True	Non-planar
SOMAS	Pfund (2001)	False	False	False	True	True	Non-planar
Cell Tuple	Pigot (1995)	False	True	False	True	True	Planar
UDM	Coors (2003)	False	True	False	False	False	Planar
3DGT	Zeitouni *et al.* (1995)	True	False	False	True	False	Both
GTP	Lixin and Wenzhong (2003)	True	False	False	False	True	Non-planar
QTPV	Gong *et al.* (2004)	True	False	False	False	True	Both
CIEL	Levy *et al.* (2001)	True	False	False	False	True	Both
(Unnamed)	Wei *et al.* (1998)	True	False	False	True	True	Non-Planar
POINCARE TEN	Penninga *et al.* (2006)	True	False	True	False		Non-Planar

Table 15 – Comparing Existing B-Rep Structures

4.7 Querying 9-Intersection Relationships from the B-Rep Structure

The identification of the 9-Intersection Relationships from a B-Rep structure depends on the identification of the primitives forming the interior, boundary and exterior of each object. These are then compared – common primitives (for example a shared Edge) imply the existence of a non-disjoint topological relationship.

4.7.1 Identifying the Interior, Boundary and Exterior of an Object using Primitives

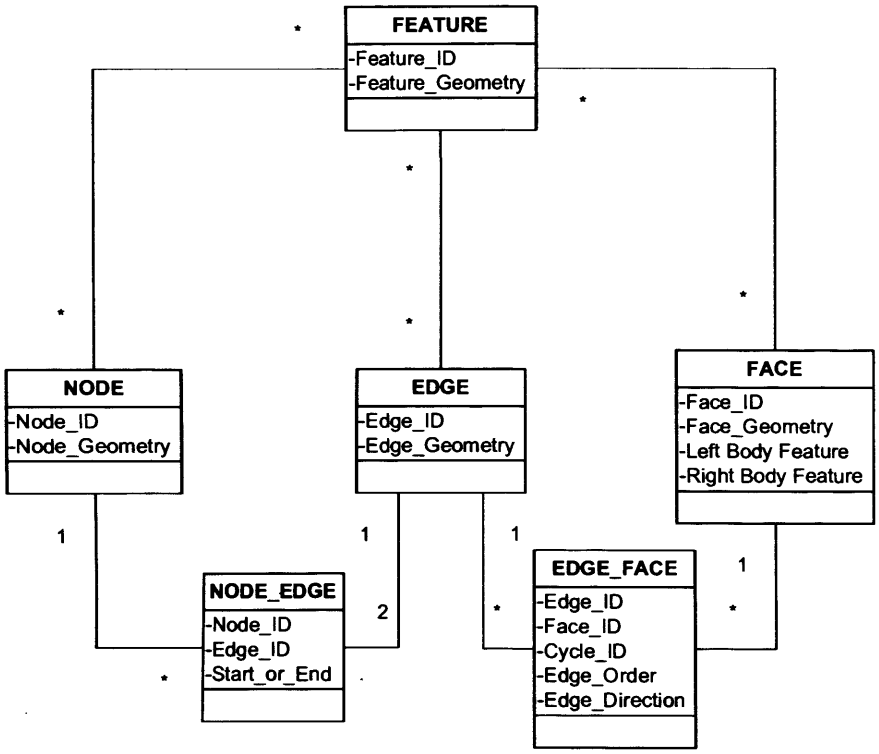


Figure 12 – B-Rep Structure

Figure 12 shows an Object-Relational representation of a basic 3D B-Rep structure (with the FACE identifying a Left and Right Body Feature). The inclusion of a generic FEATURE object representing the Point, Line, Surface and Body geometry types identifiable in a 3D setting (Molenaar, 1990) is commonly encountered within a GIS context due to the availability of spatial object types - both the original object geometry and that of the primitives is stored. This provides performance advantages for visualisation, as the object does not need to be reconstructed from the constituent primitives. Note that conventionally counter-clockwise ordering is used for the Edges around a Face, thus allowing the interior and exterior of a 3D Feature to be determined.

The identification of interior primitives in B-Rep depends on the dimension of each object. The interior of a 2D object (a surface or area) is represented by one or more 2D topological primitives (Faces). The interior of a 1D object (a line) is composed of a number of Edge primitives. The interior of a 3D Body is defined by the space enclosed by the Face primitives forming the Body. To identify the Int(A) ∩ Int(B) relationship between two surfaces A and B, the interior of each must first be identified by querying the constituent Face primitives. If the Faces reconstructing Surface A have at least one Face primitive in common with Surface B then

Int(A) ∩ Int(B) is NOT NULL. To identify the Int(A) ∩ Int(B) relationship between two 3D bodies, then a list of the Faces forming these bodies is first generated. If the shared Faces also form an enclosed space interior to both bodies, then the intersection is NOT NULL.

The boundary of an object consists of any topological primitives that have dimension less than the maximum dimension of the object as a whole (but are not contained within the object). For example, the boundary of a Body object can consist of Faces, Edges and Nodes. To identify the Bnd(A) ∩ Bnd(B) relationship between a Line A and a Body B, the boundary primitives of A must first be identified (by following joins from the FEATURE table through to the EDGE and NODE_EDGE tables). Similarly, the boundary primitives of B are identified as the FACE, EDGE and NODE objects returned by joining through from the FEATURE table to the FACE table and so forth. If the NODE primitives returned for the boundary of the line A are shared with those returned for Body B then Bnd(A) ∩ Bnd(B) is NOT NULL.

In a B-Rep structure the exterior an object is not explicitly stored – it is taken to be all the space not occupied by the interior or boundary of the object. In practice, if a primitive of A is not shared with a primitive of B then it is taken to be outside B (i.e. intersecting with the exterior of B). This is valid in most cases.

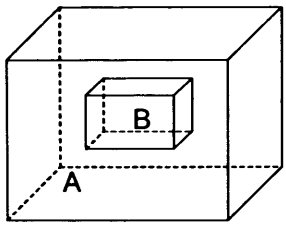


Figure 13 – Body/Body Containment – No Cavity in A

However, for the topological relationship of containment, as illustrated in Figure 13 above, this approach would return an incorrect result for the intersection of the Exterior of A and the Boundary of B (the 9-Intersection Matrix for containment is given in Table 21 below). The boundary primitives of B (Faces, Edges and Nodes) do not form part of object A. Using the above test, they would be identified as intersecting with the exterior of A. Therefore Ext(A) ∩ Bnd(B) would return NOT NULL. However, these primitives do not intersect the exterior of A but are in fact contained within A.

A number of approaches have been suggested to overcome this issue in GIS. In particular, in 2D it is possible to explicitly detail left and right Face information for Edges. Additionally, the concept of a 'Universe' polygon having ID 0 is used to represent the exterior polygon and

ensure that Edges on the perimeter of the dataset are assigned both left and right polygons. If one of the Edge primitives of a 2D area object B has a left or right Face that is not part of object A then the Ext(A) ∩ Bnd(B) relationship is TRUE. For the 1D case (two lines intersecting, embedded in 2D space), any Nodes forming part of line B and not shared with line A will return TRUE for Ext(A) ∩ Bnd(B).

Extending this to the 3D case, information detailing the left and right Body objects for each Face allows the identification of such relationships, where left or right is defined by the direction of the Edges around the Face. Again, Body ID 0 can be used to symbolize the external 'Universe' space. However, in the 3D case, left and right polygon information cannot now be stored for Edges as these may be shared between more than two Faces. Therefore the algorithm required to identify the relationship between two surfaces A and B must be adapted, so that if an Edge primitive of surface B is not shared with surface A, then Ext(A) ∩ Bnd(B) is also generally NOT NULL.

Further investigation reveals additional containment exceptions that cannot be handled using this B-Rep structure. In the case of a point object inside a Body object, no shared primitives will be returned. This is topologically correct, as the Node representing the point is not a topological primitive of the Body and cannot thus be queried through the FEATURE/FACE/EDGE/NODE hierarchy shown in Figure 12. However, this will lead to the false identification of a DISJOINT relationship between the point and Body as no primitives are listed as shared.

Illustrations of these possible containment exceptions are shown in Table 16 below, using simple objects (i.e. no holes or cavities) for both 2D and 3D embedding space. The number of containment exceptions increases when the embedding space changes from 2D to 3D. Each exception can be handled by the inclusion of a containment exception table in the B-Rep schema.

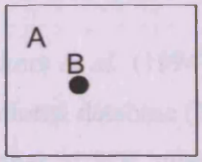
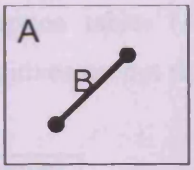
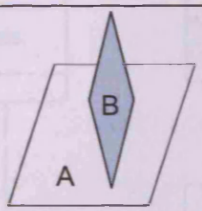
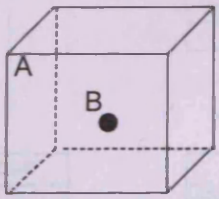
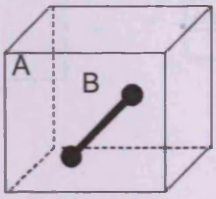
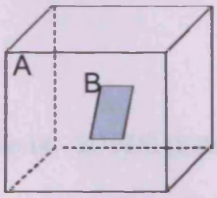
Relationship	*2D Embedding*	*3D Embedding*
	Node primitive representing B does not form part of A. Therefore use exception table.	Node primitive representing B does not form part of A. Therefore use exception table.
	Edge primitive representing B does not form part of A. Relationship can be determined as the Face representing A is both left and right of the Edge representing B.	Edge primitive representing B does not form part of A. Exception table required as Edge may be shared by more than two Faces in 3D i.e. a left/right relationship for Edges cannot be determined.
	N/A	Edge of B intersecting with A does not form part of A. Exception table required (as above).
	N/A	Node representing B does not form part of A. Exception table required.
	N/A	Edge representing B does not form part of A. Exception table required.
	N/A	Face representing B does not form part of A. However, the relationship can be determined as the left and right bodies for the Face are both the same.

Table 16 - Containment Exceptions in 2D and 3D Embedding Space

4.8 A Typical 3D GIS B-Rep Structure - 3DFDS

A 3D GIS implementation of a B-Rep structure is presented here - the 3D Formal Data Structure (3DFDS, Molenaar 1990). 3DFDS is perhaps the best known of the 3D GIS B-Rep topological data structures, and has been implemented or enhanced by a number of authors, including Rikkers *et al.* (1994), Zlatanova (2000), Tempfli (1998) and Coors (2003). The basic components of this structure are the NODE, ARC and FACE primitives, along with SURFACE, BODY, LINE and POINT object types. Containment exceptions are modelled, through the use

of the ARC_IN_BODY, ARC_ON_FACE, NODE_ON_FACE and NODE_IN_BODY relationships.

Rikkers *et al.* (1994) provide details of an implementation of Molenaar's (1992) 3DFDS in a relational database (Figure 14) and illustrate the tables required to create the model, as well as a number of SQL queries required to query the model. Relationships between the containment exception tables (ARCONF, ARCINB, NODEINB, and NODEONF) and the topological primitives are not shown to maintain diagram clarity.

Figure 14 - 3DFDS (Rikkers *et al.*, 1994)

Molenaar (1992) imposes a number of restrictions on 3DFDS. These are listed in the form of conventions, and constrain the types of objects that can be used within the structure to elementary objects – i.e. those with lines and surfaces that do not self-intersect and which do not have unconnected parts. The Rikkers *et al.* (1994) implementation of 3DFDS was carried out within a relational database and does not take account of the spatial object data types available in an Object-Relational setting.

4.9 Validating the 3DFDS Approach

Given the requirement to incorporate topological functionality into 3D GIS and to implement the mapped 9-Intersection relationships within this environment, 3DFDS is reviewed in terms of:

- The support provided for real-world 3D data.
- The support provided for the 3D 9-Intersection relationships identified by Zlatanova (2000).

Following on from this review, an extended version of 3DFDS is presented to overcome some of the issues encountered, and to incorporate Object-Relational spatial data types.

4.9.1 Handling 3D Objects

4.9.1.1 Complex Objects

Complex objects, i.e. those containing cavities (such as that shown in Figure 15), tunnels or holes, are directly supported by the 3DFDS. Internal cavities in 3D bodies are modelled by identifying the left and right bodies for each Face. Molenaar (1992) states that "for the sake of completeness the outer space is also a Body." This Body is given ID 0.

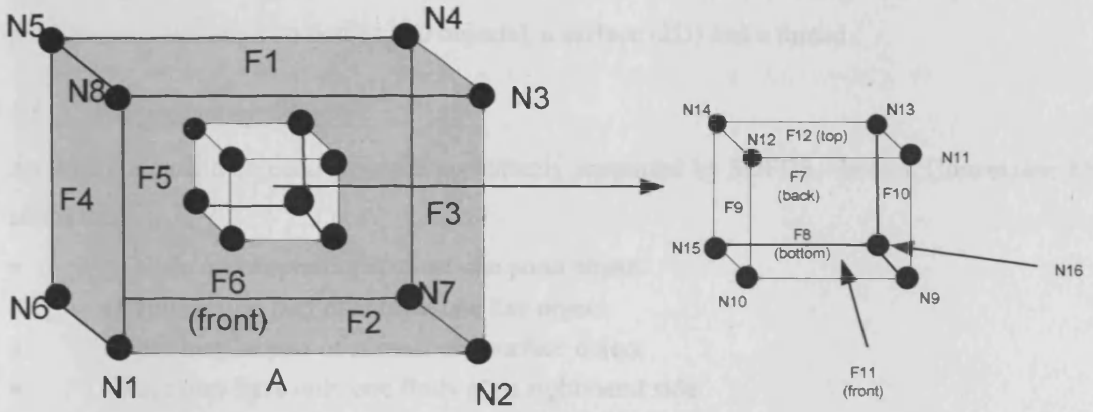


Figure 15 - Body with Internal Cavity, showing Topological Primitives

Thus the FACE table for the Body A in Figure 15 above will contain the following entries.

FID (Feature ID)	*FPARTOFS (Face Part of Surface)*	*BIDLEFT (Body ID Left)*	*BIDRIGHT (Body ID Right)*	*TEXTUREF (Reference to the Display Texture)*
F1	NULL	0	A	NULL
F2	NULL	0	A	NULL
....				
F6	NULL	0	A	NULL
F7	NULL	A	0	NULL
F8	NULL	A	0	NULL
F9	NULL	A	0	NULL
....				
F12	NULL	A	0	NULL

Table 17 - FACE Table for Body with Internal Cavity

4.9.1.2 Compound Objects

Compound (multi-part) objects are not directly supported by 3DFDS, which distinguishes between four simple object types – Point, Line, Surface and Body. However, it is possible to sub-divide compound objects into simple objects, extending the relational structure to link these objects together at a higher level in the structure. Molenaar (1992) illustrates this process with an example involving two bodies (3D objects), a surface (2D) and a thread.

4.9.1.3 Overlapping Objects

As with compound objects, these are not directly supported by 3DFDS. In fact, Convention 11 states that:

- "a Node may represent at most one point object
- an Arc may be part of at most one line object
- a Face may be part of at most one surface object
- a Face may have only one Body at its right-hand side
- a Face may have only one Body at its left hand side"

De Hoop *et al.* (1993) describe an alteration to the 2D Formal Data Structure (which forms the foundation of 3DFDS) to create multi-valued vector maps, required to support data layer overlay in 2D GIS. They evaluate options for implementation and select an approach where one set of common topological primitives is created and related to multiple layers of vector data. Although the concept of layers is less well defined in 3D, a similar approach is described for 3DFDS below (Section 4.9.3) – this also ensures backwards compatibility with 2D datasets.

4.9.1.4 Curved Surfaces

In the context of analytical topology (i.e. querying binary relationships), without taking into consideration data maintenance or the topological engine, the requirement to model curved lines and curved surfaces does not exist. Two 3D objects are adjacent whether the surface where they

touch is curved or planar. Convention 7 in 3DFDS states that "Faces are planar (unless explicitly described otherwise, Faces should not be self-intersecting)". Thus curved lines and surfaces are not directly represented in 3DFDS. However, a series of planar Faces may be used to represent the curved surfaces. A process of triangulation may provide the most appropriate representation, or alternatives such as that shown in Figure 16 considered, depending on the nature of the dataset and the functionality available within the topological engine. No modifications to 3DFDS are required to support this process.

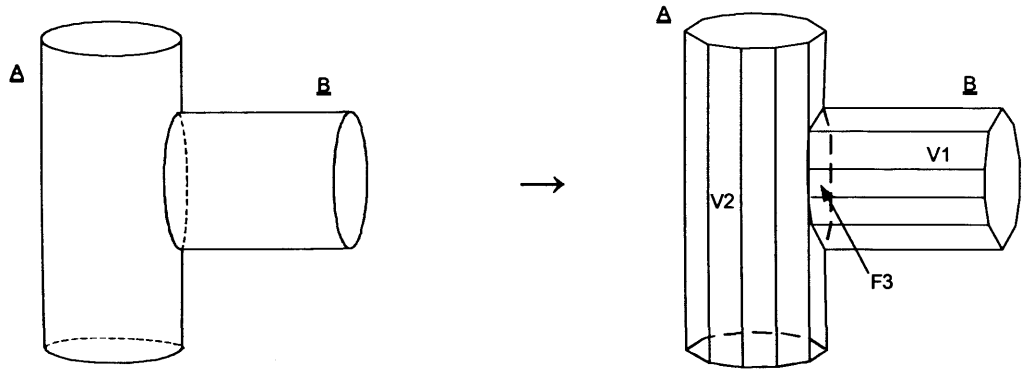


Figure 16 - Decomposition of Curved Surfaces into Planar Primitives

4.9.1.5 *Non-Manifold Objects*

Non-manifold objects are problematic in terms of determining the inside and outside of an object, the left and right sides of surfaces and hence identifying any containment relationships between objects or which Body is on which side of each Face. Hoffman (1989, pg. 61) suggests that these objects are handled by breaking them down into manifold objects. Using this approach, it is possible to represent non-manifold objects directly in 3DFDS, as shown in Figure 17 below.

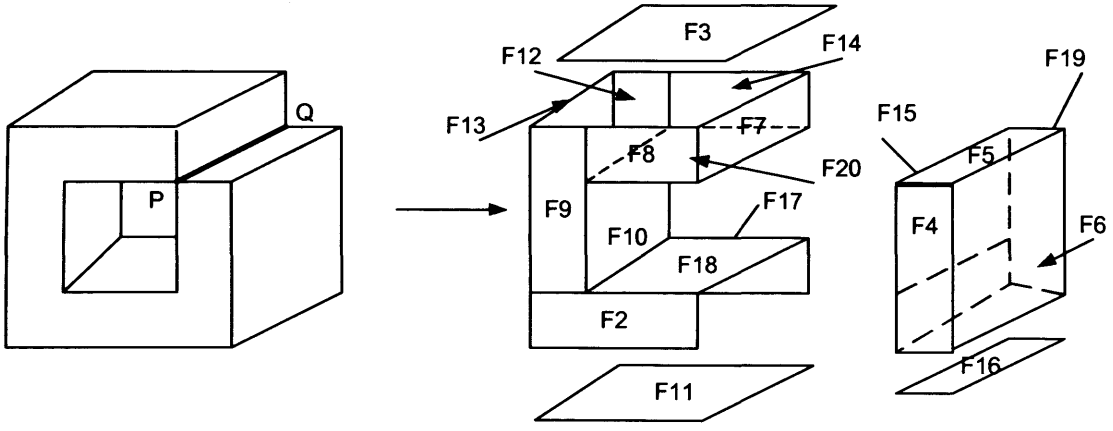


Figure 17 – B-Rep of Non-Manifold Objects

In the case of Figure 17, Edge PQ is associated with four Faces of the object – F7, F20, F5 and F15, thus identifying the object as non-manifold surfaces. This can be modelled in the EDGE

table, resulting in four entries associated with Edge PQ, which does not violate any 3DFDS conventions. However, given the dimension of Edge PQ, it will be automatically assumed that this represents a boundary primitive of the Object as it has dimension less than that of the object itself (although this may not be correct). Thus although such objects can be stored in 3DFDS, binary topological query results obtained may be incorrect.

4.9.2 Support for 9-Intersection Relationships

A number of examples illustrating the range of relationships supported and not supported by 3DFDS are given here.

4.9.2.1 Example 1 – Adjacent Bodies – R287

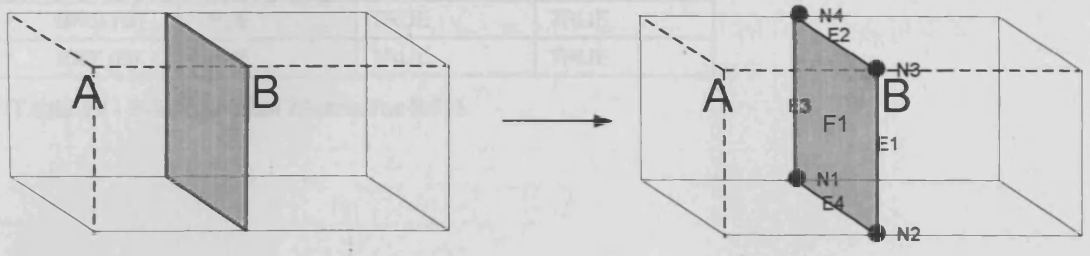


Figure 18 - Body/Body Adjacency - R287

	INT (A)	*BND (A)*	*EXT (A)*
INT (B)	FALSE	FALSE	TRUE
BND (B)	FALSE	TRUE	TRUE
EXT (B)	TRUE	TRUE	TRUE

Table 18 - 9-Intersection Matrix for R287

In Figure 18 (for which the corresponding 9-Intersection matrix is given in Table 18), the fact that Face F1 is shared, has Body A on one side and Body B on the other and forms a boundary of both A and B can be determined – A and B are both 3D bodies, thus all associated primitives with dimension 2 or less are boundary primitives. This implies that $\partial A \bigcap \partial B$ is TRUE (i.e. NOT NULL). The $A^0 \bigcap B^0$ relationship can be determined from the fact that only one Face is shared between the bodies, and is FALSE. The intersections of the exterior of the objects are determined by examining the non-shared primitives. By default, a primitive of A that does not intersect with a primitive of B is exterior to B. This relationship is thus directly supported by 3DFDS.

4.9.2.2 Example 2 – Surface/Surface Overlap – R511

In Figure 19, Object A (a surface) shares a Face (F2) with Object B (also a surface). As both objects are 2D, then the shared Face represents the $A^0 \bigcap B^0$ relationship, which will return a NOT NULL intersection. Convention 11 of 3DFDS states, however, that "a Face may be part

of at most one surface object" (Molenaar 1992). Thus this situation is in violation of Convention 11, as Face F2 represents both Surface A and Surface B.

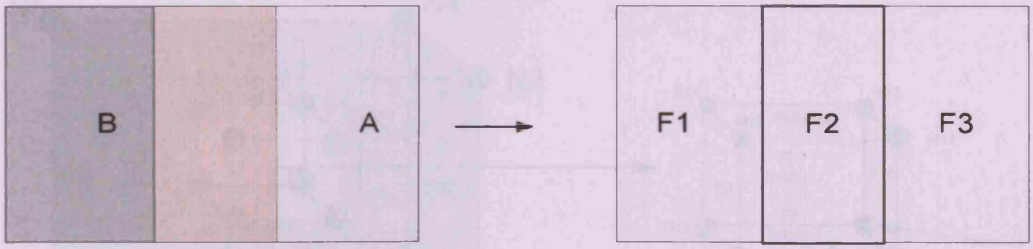


Figure 19 - Surface/Surface Overlap - R511

	INT (A)	*BND (A)*	*EXT (A)*
INT (B)	TRUE	TRUE	TRUE
BND (B)	TRUE	TRUE	TRUE
EXT (B)	TRUE	TRUE	TRUE

Table 19 - 9-Intersection Matrix for R511

4.9.2.3 *Example 3 - Body/Body Touch – With Cavity – R287*

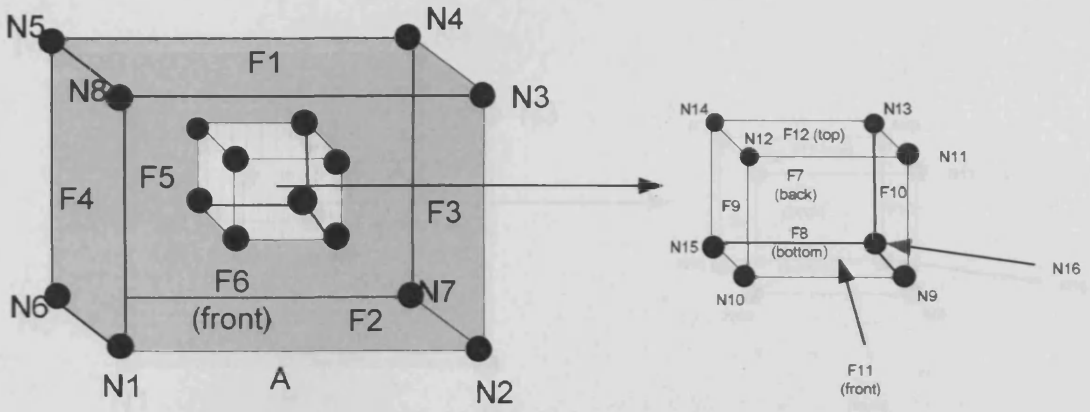


Figure 20 - Body/Body Touch - Body A has an internal cavity which surrounds Body B

	INT (A)	*BND (A)*	*EXT (A)*
INT (B)	FALSE	FALSE	TRUE
BND (B)	FALSE	TRUE	TRUE
EXT (B)	TRUE	TRUE	TRUE

Table 20 - 9-Intersection Matrix for R287

In Figure 20 (for which the corresponding 9-Intersection matrix is given in Table 20), assume that Face F9 is bounded on the left by Body A (the outer Body) and on the right by Body B (the inner Body). Thus this relationship does not violate Convention 11 of 3DFDS, which states that "a Face may have only one Body at its right-hand side" and "a Face may have only one Body at its left hand side". The R-value for the relationship can thus be determined by querying 3DFDS for shared primitives.

4.9.2.4 *Example 4 - Body/Body Containment – No Cavity - R220*

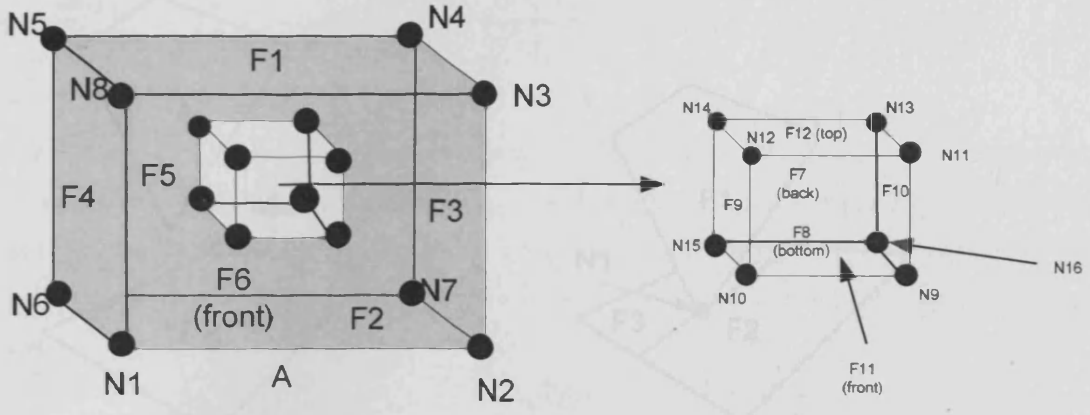


Figure 21 - Body/Body Containment - No Cavity – R220

	INT (A)	*BND (A)*	*EXT (A)*
INT (B)	TRUE	FALSE	FALSE
BND (B)	TRUE	FALSE	FALSE
EXT (B)	TRUE	TRUE	TRUE

Table 21 - 9-Intersection Matrix for R220

Convention 11 of 3DFDS states that "a Face may have only one Body at its right-hand side" and "a Face may have only one Body at its left hand side". For example Face F9 in Figure 21 above, Body A is on the left hand side. However, as Body A does not contain a cavity, both Body A and Body B are on the right hand side of F9, violating this convention. This relationship is thus not supported by 3DFDS (note that this issue has been addressed in 2D by De Hoop *et al.* 1993 – a similar modification will be made to 3DFDS, see Section 4.9.3).

4.9.2.5 *Example 5 – Surface/Surface Touch – R063*

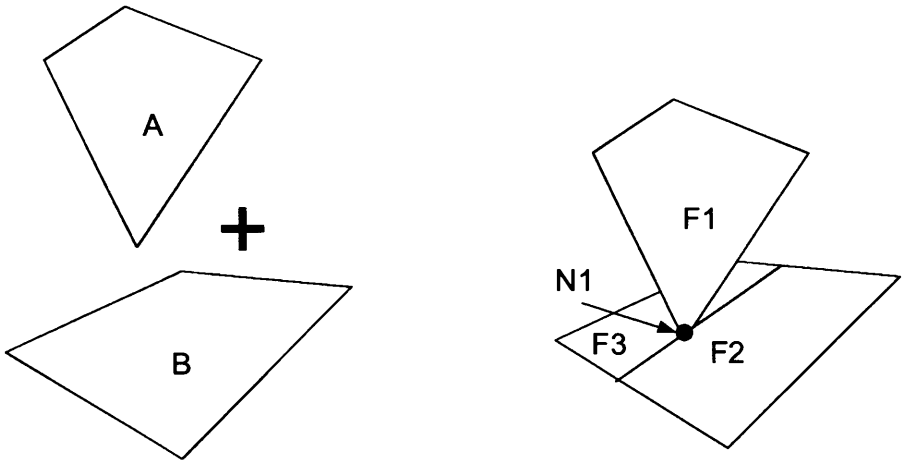


Figure 22 - Surface/Surface Touch - Simple Surfaces

	INT (A)	***BND (A)***	***EXT (A)***
INT (B)	FALSE	TRUE	TRUE
BND (B)	FALSE	FALSE	TRUE
EXT (B)	TRUE	TRUE	TRUE

Table 22 - 9-Intersection Matrix for R063

In this case, Convention 8 of 3DFDS, which states that "Faces can be topologically related to each other only through their boundaries, not though their interiors, i.e. two Faces should not intersect or the border of one Face should not touch the interior of another Face", is violated. Therefore, 3DFDS requires surface B is split into Faces F2 and F3 to ensure that the Face representing surface A (F1) does not intersect the interior of any other Faces. No entry is required in the NODEINF exception table, as Node N1 does not form part of surface B.

4.9.2.6 *Example 6 – Surface/Surface Touch – R287*

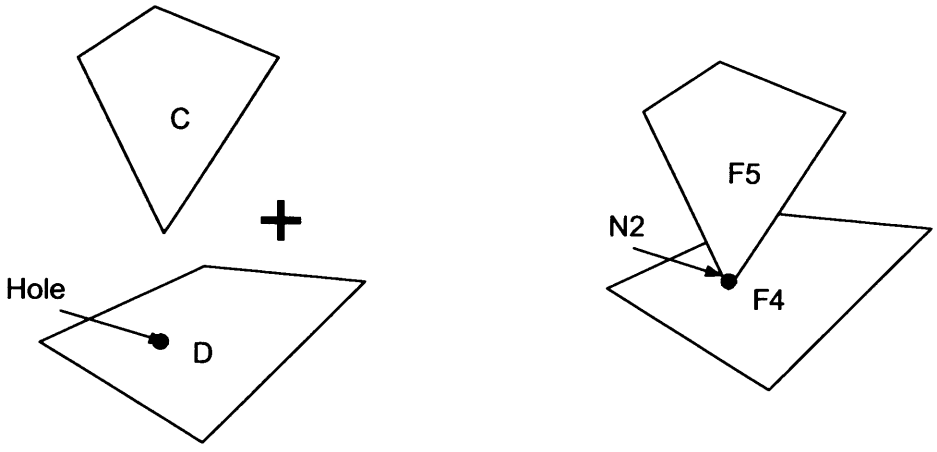


Figure 23 - Surface-Surface Touch –Hole in Surface

	INT (C)	*BND (C)*	*EXT (C)*
INT (D)	FALSE	FALSE	TRUE
BND (D)	FALSE	TRUE	TRUE
EXT (D)	TRUE	TRUE	TRUE

Table 23 - 9-Intersection Matrix for R287

Figure 22 can be contrasted with the relationship R287, illustrated in Figure 23. In the latter case, a hole exists in Surface D, which forms part of the boundary of this surface. Therefore an entry in the NODEINF exception table is created for the Node representing this pre-existing hole. Convention 8 is not violated for these objects – Node N2 forms part of the boundary of Face F4.

4.9.2.7 *Example 7 – Body/Line Containment – R220*

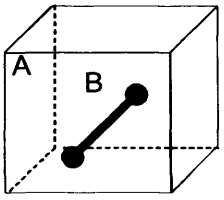


Figure 24 - Line B contained within Body A

	INT (A)	*BND (A)*	*EXT (A)*
INT (B)	TRUE	FALSE	FALSE
BND (B)	TRUE	FALSE	FALSE
EXT (B)	TRUE	TRUE	TRUE

Table 24 - 9-Intersection Matrix for R220

Figure 24 (and the corresponding 9-Intersection matrix Table 24) above illustrates the situation where Line B is contained within Body A. The containment relationship is represented in 3DFDS through the use of the ARCINB exception table. Thus, in addition to identifying the interior and boundary primitives of Objects A and B and validating their intersection, each containment exception table must be queried – in this case, the link between the Arc representing B and Body A implies that the $A^0 \cap B^0$, $A^0 \cap \partial B$ and $A^0 \cap B^-$ relationships are all TRUE – even though there are no primitives shared between A and B.

4.9.3 Extended 3DFDS

A number of the characteristics of 3D objects and a number of 3D 9-Intersection relationships identified by Zlatanova (2000) cannot be supported by 3DFDS as defined by Molenaar (1992) and implemented by Rikkers *et al.* (1994). However, extensions can be made to the structure to overcome the issues described. Additionally, due to the emergence of Object-Relational databases, spatial representations of the Node, Edge and Face primitives can also be included in the structure.

4.9.3.1 *Extension 1 – Object-Relational Representation of the Primitive Geometries*

The introduction of Object-Relational databases permits the addition of spatial objects to represent the NODE, ARC and FACE primitives. Point, Line, Surface and Body objects are also merged as the Object-Relational spatial data type allows multiple object types to be represented. This may facilitate data maintenance as, for example, an ordered list of coordinates making up a Face can be easily extracted (see Section 5.10.2 for further discussion of the advantages and disadvantages of this approach).

4.9.3.2 *Extension 2- Support for Compound Objects*

This is required to support multi-part (compound) objects. As described above, these can be represented in 3DFDS by sub-dividing the compound object into simple objects, using a table known as a TOPO_PART_TABLE to link the simple parts to the parent object.

4.9.3.3 *Extension 3 – Many:Many Relationship between Surface and Face*

Convention 11 of 3DFDS states that "a Face may be part of at most one surface object" (Molenaar 1992). However, Figure 19 above illustrates a situation where this does not hold true – i.e. where two surface objects overlap. A join table, to represent the many:many relationship between the Face primitives and the parent object is required. This is known as TOPO_FACE.

4.9.3.4 *Extension 4 – Many:Many Relationship between Point and Node*

Convention 11 of 3DFDS states that "a Node may be part of at most one point object" (Molenaar 1992). As overlapping points are commonly encountered in 3D datasets, a join table, to represent the many:many relationship between the Node primitives and the parent object is required. This is known as TOPO_NODE.

4.9.3.5 *Extension 5- Requirement for a Volume Primitive*

B-Rep structures do not explicitly represent the interior of a 3D Body. This is required to support rapid performance of the determination of any 9-Intersection relationships involving two Body objects. Although the interior of a Body can be reconstructed from its constituent Faces using the left and right Body information, a number of situations arise where this approach is insufficient for relationship determination. Table 25 shows left and right Body values for shared Faces for each 9-Intersection Body/Body relationship.

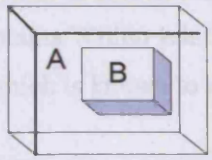
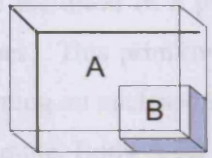
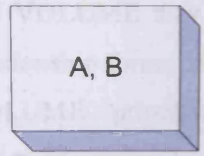
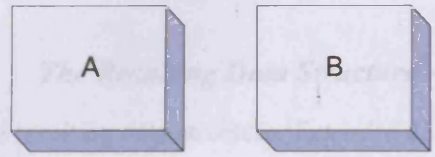
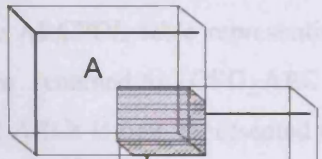
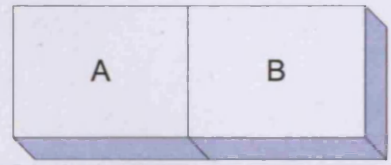
Illustration (Body A is outside, Bodies do not contain cavities)	*Relationship*	*Relevant Faces*	*Left Body*	*Right Body*
	Contains/ Within	Shared Faces	A, B	A
	Covers/ Covered By	Shared Faces, Bounding A Shared Faces, Internal to A	A, B A, B	0 A,B
	Equals	All Faces	A, B	0
	Disjoint	All Faces	N/A	N/A
	Overlap	Shared Faces	A, B	A, B
	Meet	Shared Face	A	B

Table 25 - 9-Intersection Relationships between Simple Bodies

All relationships apart from DISJOINT and MEET violate Convention 11 of 3DFDS, as one or more Faces are related to more than one Body on either side. This can be overcome by creating an additional join table to represent the many:many relationship between the FEATURE object and the FACE primitive.

Taking the Overlap relationship, and using this new join relationship, six shared Faces can now be identified between Body A and Body B (shown hatched in the Overlap relationship in Table 25 above). However, there is no means of determining, without using computational geometry

algorithms, that these shared Faces in fact form an enclosed space having the same dimension as the objects in question, thus representing an intersection of the interior of these objects. Similar issues arise when considering the Covers/Covered By relationship (although the Contains/Within relationship can now be identified, as all the shared Faces also make up Body B which is known to enclose a 3D space).

The inclusion of a primitive to explicitly represent the interior of a 3D Body resolves these issues. This primitive, known as VOLUME primitive, allows a set of Faces to be marked as forming an enclosed space. One FEATURE may be related to many VOLUME primitives (for example, Body A for the Overlap relationship in Table 25 is comprised to two VOLUMEs) and one VOLUME may form part of many FEATURES (the shared VOLUME in this Overlap relationship forms part of both Body A and Body B). Each FACE is now related to at most two VOLUME primitives. The Volume primitive allows direct determination of the $A^0 \cap B^0$ relationship for two 3D Bodies.

4.10 The Resulting Data Structure – Extended 3DFDS

The resulting data structure, *Extended 3DFDS*, is shown in Figure 25. To simplify the diagram, the relationships between the exception tables and the topological primitives are not shown. The ARCPOL table representing the many:many relationship between objects and Edges has been renamed to TOPO_ARC for consistency. The many:many relationship between NODEs and ARCs is now represented by the NODE_ARC join table. The ARCINB and NODEINB tables have now been renamed to ARCINV and NODEINV due to the introduction of the VOLUME primitive.

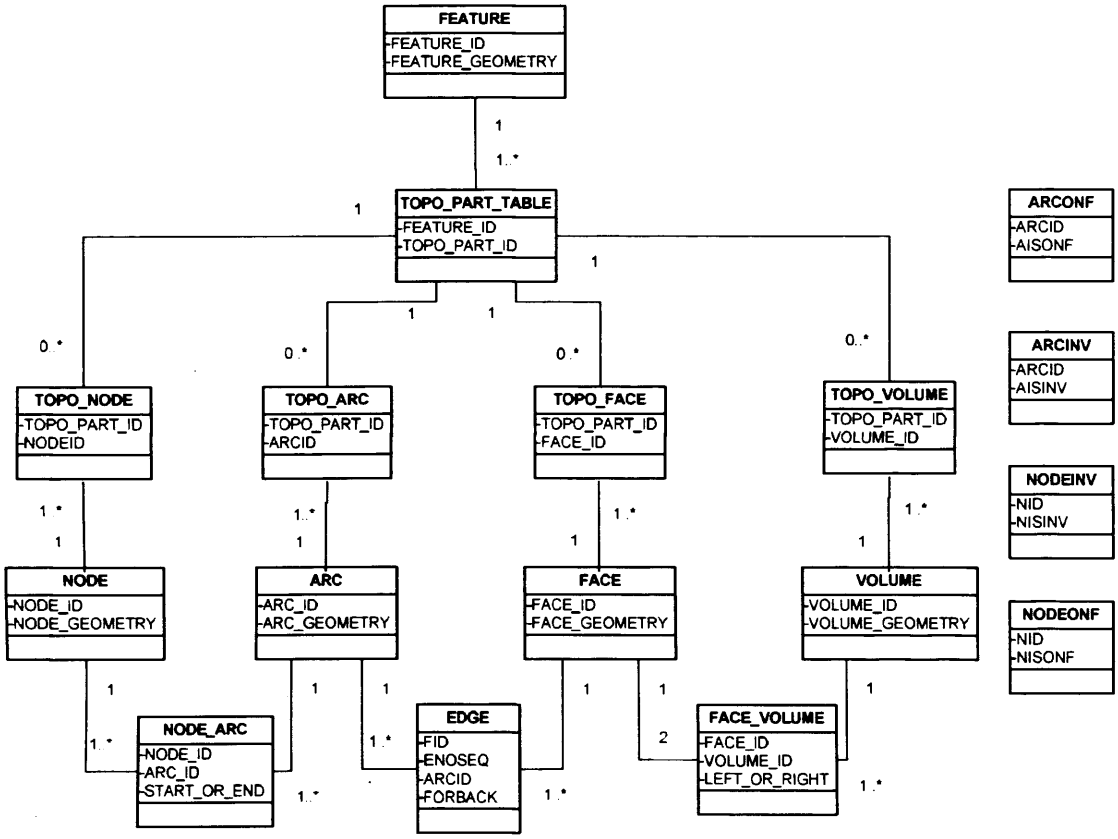


Figure 25 - Extended 3DFDS

4.10.1 Strengths of Extended 3DFDS

The extended 3DFDS data structure shown in Figure 25 overcomes the limitations of the basic 3DFDS, in particular allowing for multiple objects to share primitives and for Body/Body containment and covers relationships to be modelled. The structure can now model the 9-Intersection relationships in 3D identified by Zlatanova (2000).

The use of the spatial object data type to incorporate the Point, Line, Surface and Body classes facilitates visualisation of objects in a 3D setting – there is no need to build the object from the primitives for visualisation purposes. Similarly, spatial objects are used to represent the primitives, allowing these to be visualised and edited separately if required (assuming that appropriate triggers are in place to ensure that data is maintained correctly). B-Rep structures are well understood, and 3DFDS extends topological structures implemented in a number of commercial 2D GIS, providing backwards compatibility.

4.10.2 Limitations of Extended 3DFDS

Both 3DFDS and Extended 3DFDS implement an ARCONF containment exception table to handle Arcs contained on the surface of a Face. This is to overcome the fact that an Arc can

now be linked to more than two Faces. However, it is also possible to identify this containment relationship by linking the Arc to the Face twice (i.e. having two records in the EDGE table) if manifold objects are assumed. This approach would be consistent with that taken for Faces in Volumes, which are identified by the fact that two records for the Face (one with the Volume on the Left and one on the Right) appear in the FACE_VOLUME table.

Convention 7 in 3DFDS and Extended 3DFDS mandates planar Faces. In terms of the ability to model real-world objects this approach is perhaps rather limiting. The suggested approach - breaking down the surface into a series of planar Faces (such as TIN triangles) - has issues in terms of the number of associated internal Edges required to build up the Faces. Conceptually, it is the boundary Edges of the Face that are important in terms of relationship specification (along with any contained Edges stored in the exception table). Thus, extending 3DFDS to support other representations of Non-Planar Faces should also be considered. This will reduce the storage requirements, although it may not then be possible to rebuild the Face from the Edge primitives (see Section 4.2 for possible representations of curved surfaces, and Table 76 for details of the impact of these types of objects on binary relationship query performance).

Implementation of Extended 3DFDS requires additional tables when compared to basic 3DFDS. This in turn complicates the data query, population and maintenance processes. In relation to binary topological query performance, the main focus of this research, examining both the basic and the extended 3DFDS B-Rep structure identifies two additional issues which may impact query performance.

Issue 1 – Number of Relational Joins Required

The algorithm to determine the 9-Intersection relationship between two objects initially involves determining the dimension of each object. Once this is done, primitives are flagged as interior or boundary primitives. Finally, the algorithm determines the intersection between each group of primitives (see Chapter 7 for a detailed description of this algorithm). For 3DFDS and Extended 3DFDS, the queries involved in the second step of this process require a relatively high number of join relationships to be followed. For example, to determine the Node primitives associated with a Body object, the query must follow joins from FEATURE to TOPO_PART_TABLE, from TOPO_PART_TABLE to TOPO_VOLUME, from TOPO_VOLUME to FACE_VOLUME, from FACE_VOLUME to EDGE_FACE and finally from EDGE_FACE to NODE_EDGE. Similar (although more compact) queries are required to determine constituent Nodes for surface and line objects.

Atzeni *et al.* (1999, page 337) note that relational joins are one of the more costly operations in terms of query performance. Joins take two candidate sets of objects and attempt to find matches according to the criteria given in the query – their potentially high costs lies in the possibility of large numbers of tuples being considered as potential candidates on either side of the join, greatly increasing the number of comparisons that are required. Algorithms for join operations are described in Table 111, Appendix 7. Although it may be possible to de-normalise the structure to ameliorate this issue (for example by assuming that a Face will only ever be shared by 2 Volumes or that an Arc will only have two end Nodes) this will restrict the range of object types that the structure can model. Section 5.9.3 gives a discussion of an Object-Reference implementation, which may also mitigate join performance issues.

Issue 2 – Exception Tables

In both 3DFDS and Extended 3DFDS, four containment exception tables must be queried before all shared primitives are identified for each pair of objects. This complicates the queries to identify shared primitives, as well as adding to the number of joins to be traversed.

4.11 Summary

This Chapter first outlined characteristics of 3D objects that add to the complexity of the algorithms for identifying binary topological relationships, including the presence of cavities, multi-part objects and non-manifold surfaces and presented an example of one such algorithm. Two approaches to binary relationship implementation, As-Required and structure-based, were then described and compared. It was noted that even in 2D GIS no consensus has been reached as to which of these is optimal.

Structure based approaches were reviewed, and B-Rep identified as appropriate for implementation within the context of this research. A typical B-Rep structure in a GIS context, 3DFDS, was then considered in some detail, both in terms of its ability to model the identified characteristics of 3D objects and in terms of the processes and queries required to extract 9-Intersection binary topological relationship information from the structure. A number of extensions to the structure were proposed in order to overcome limitations in this regard. Two issues were identified for the resulting Extended 3DFDS - the high number of join traversal operations required and the presence of four exception tables. An alternate structure, designed to overcome these issues, is detailed in Chapter 5.

5 Simplified Topological Structure

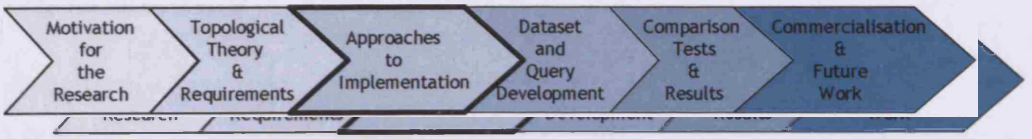


Figure 26 - Overview of Document Structure showing Context of this Chapter

5.1 Introduction

Chapter 4 outlined two limitations of the Extended version of 3DFDS in relation to 9-Intersection query performance, namely the number of relational join queries required to identify primitives relating to objects (in particular Node primitives of Body objects) and the requirement for containment exception tables to handle cases not modelled by the Node/Edge/Face/Volume hierarchy. Taking these issues into account, this Chapter proposes an alternative structure, known as the Simplified Topological Structure (STS), designed to improve query performance. This structure was developed as part of this thesis. A conceptual model for the structure is first presented, with a logical implementation of this model also described, taking into account the availability of Object-Relational databases. The structure is reviewed in terms of its ability to handle the various characteristics of 3D objects outlined in Chapter 4, including compound objects, containment, complex objects and non-manifold objects. Strengths and weaknesses of STS are examined, and the structure is compared to the ISO 19107 and Extended 3DFDS.

STS has been designed to optimise query performance for the identification of 9-Intersection topological relationships. Additionally, the availability of Object-Relational databases and hence the ability to model spatial objects was also a consideration of the design process. The following design description is presented within this context.

5.2 *Conceptual Model*

Figure 27 gives an overview of the Simplified Topological Structure.

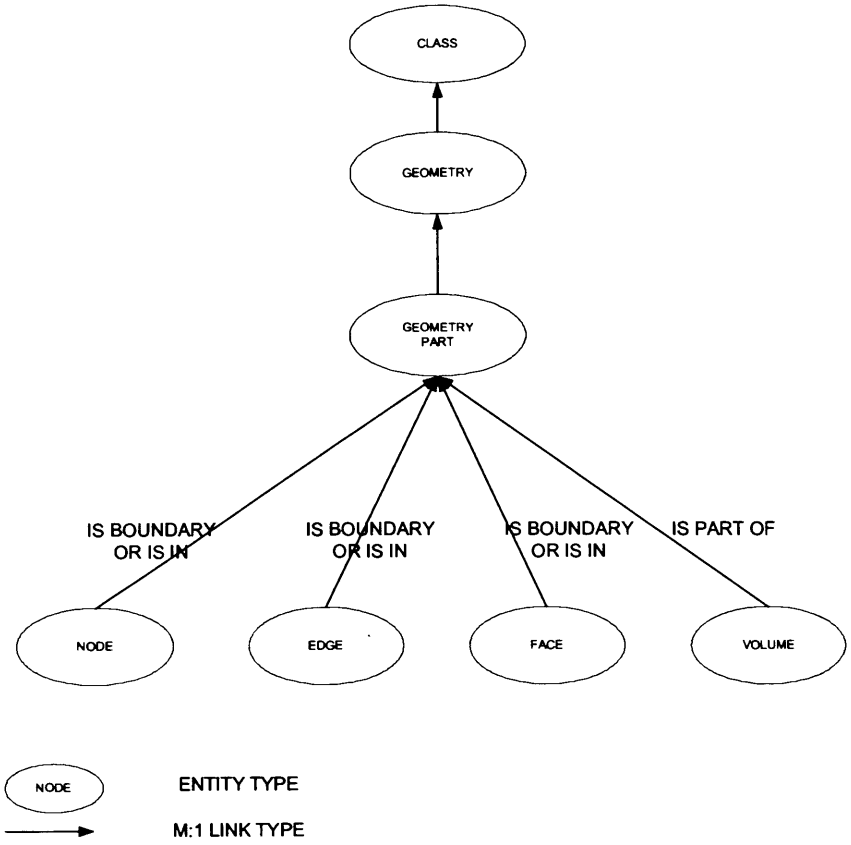


Figure 27 - The Simplified Topological Structure

The conceptual model for STS is presented in three parts. Firstly, a definition of the geometry of the objects that are supported by this structure is given. This is followed, in Part II, by a description of the topological primitives. Part III presents a diagram linking the primitives to the object geometry types, and hence the objects to their corresponding topology.

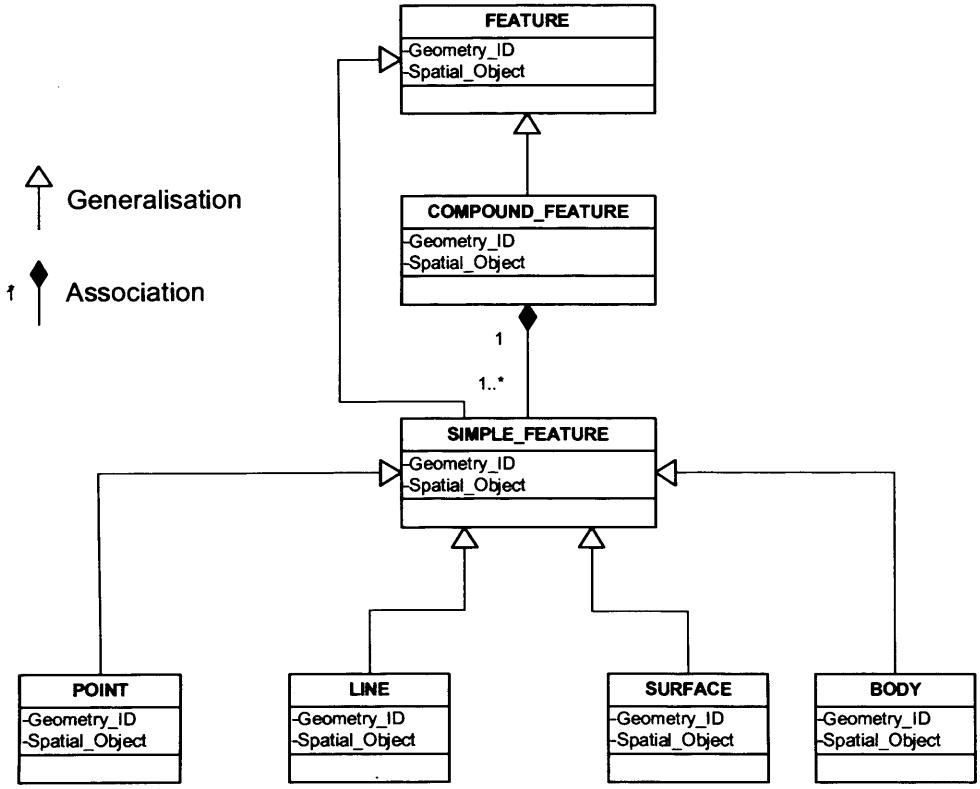


Figure 28 – STS Geometry Model – Conceptual

5.2.1 Part I Geometry

Figure 28 presents classes related to the geometry component of the conceptual model of STS. A generic parent FEATURE class, where objects may be composed of multiple parts and/or may be complex (contain cavities, tunnels or holes), forms the root of the model. Two subclasses are defined – simple FEATURE (i.e. single part, having one dimension) and compound FEATURE (consisting of more than one simple geometry, where the individual parts may have different dimensions and may be disconnected from each-other in space). Both of these may contain cavities and holes. The final components of this model are the Point, Line, Surface and Body objects. These are defined below.

5.2.1.1 Point

A Point is a zero-dimensional object which has a position in space but no spatial extension or length – it is defined by its location, which is represented as single tuple of 3D coordinates. A Point does not have to be unique in location – many Points may occupy identical positions in space. Compound Points may be formed from multiple disconnected Points.

Defining the Interior, Boundary and Exterior of the Point

To support the identification of 9-Intersection relationships, the interior, boundary and exterior of a Point must also be defined. The *interior* of a Point denoted by P° is the empty set. The *boundary* of the Point, denoted by ∂P, is the Point by itself. The *closure* of a Point, denoted by $\overline{P}$, is the union of the boundary and the interior, i.e. $P = \partial P \cup P^{\circ}$. The *exterior* of a Point, denoted by P^{-}, is the difference between the universe U and the closure of $\overline{P}$, *i.e.* $P^{-} = U - P$.

5.2.1.2 *Line*

A Line is a one-dimensional object, having length as a measurable extent, but no area. The shape of a Line can be defined by an ordered list of 3D coordinate tuples (which share geometric characteristics with Points, although they are not necessarily Point objects), or may be defined through other means such as mathematical equations. Lines may be straight or curved, and may self-intersect, provided that this does not preclude the identification of their interior and boundary by the topological engine. A Line does not have to be unique in location – many Lines may be superimposed. Compound Lines may be formed from multiple, disconnected parts.

Defining the Interior, Boundary and Exterior of the Line

To support the identification of 9-Intersection relationships, the interior, boundary and exterior of the Line must be defined. Whether a Line is curved or straight, the coordinate tuples that define the end Points of the Line define the boundary of the Line – in other words, the *boundary* of the Line, denoted by ∂L, is defined as the extremities (end-Points) of the Line. Note that a closed Line does not have a boundary. The interior of the Line, denoted by L^0 is defined as the link between these extremities. The *exterior* of a Line, denoted by L^{-}, is the difference between the universe U and the union of the interior and boundary of the Line, *i.e.* $L^{-} = U - (\partial L \cup L^0)$.

5.2.1.3 *Surface*

Surface objects in 3D are comparable to area objects in 2D, but can be 2D or 2.5D. Their measurable extents include area and perimeter. They are bordered by one or more closed linear elements (note that these are not necessarily Line objects, although they have identical geometric characteristics). Surfaces may be planar, may consist of multiple planar components or may be curved. The boundary of the Surfaces may also be curved. Surfaces may self-intersect, and may have internal holes. Compound Surfaces may also be formed from multiple disconnected parts which can be broken down into a number of simple Surfaces. A Surface does not have to be unique in location – Surfaces may overlap.

Defining the Interior, Boundary and Exterior of the Surface

The boundary of the Surface is denoted by ∂S, and is defined as the extremities of the Surface. Closed Surfaces do not have boundaries. The interior of the Surface, denoted by S^0, is the 2D or 2.5D space enclosed by these extremities. The *exterior* of a Surface, denoted by S^-, is the difference between the universe U and the union of the interior and boundary of the Surface, *i.e.* $S^- = U - (\partial S \cup S^0)$.

5.2.1.4 *Body*

Body objects have measurable extents including volume and surface area. They are bordered by one or more closed Surface elements (note that these are not necessarily Surface objects in their own right, although they share identical geometric characteristics). A compound Body may be formed from multiple disconnected parts, which can be broken down into many simple bodies. A simple Body may only have one outer Surface but may have several inner Surfaces, representing cavities within the Body. These Surfaces are such that they enclose a continuous space - all parts of the space can be reached from all other parts without going outside the simple Body. A Body does not have to be unique in location – bodies may overlap if required.

Defining the Interior, Boundary and Exterior of the Body

The boundary of a Body, denoted by ∂B, is the 2D or 2.5D extremities of the Body, with the interior of the Body, denoted by B^0, consisting of the 3D space enclosed by the Body. The *exterior* of a Body, denoted by B^-, is the difference between the universe U and the union of the interior and boundary of the Body, *i.e.* $B^- = U - (\partial B \cup B^0)$.

5.2.2 Part II - Topology

Figure 29 shows the topological conceptual model for the STS, along with a link between the topology and geometry. One object is related to many topological primitives and vice versa. In turn, topological primitives are sub-divided into two groups – boundary primitives and interior primitives. In 3D, Node, Edge and Face primitives can be either boundary or interior. Volume primitives cannot be boundary primitives. Optionally (denoted by dashed Lines in Figure 29) an Edge can be defined in terms of Nodes, a Face in terms of Edges and a Volume in terms of Faces. This differs from standard B-Rep, where these relationships are mandatory.

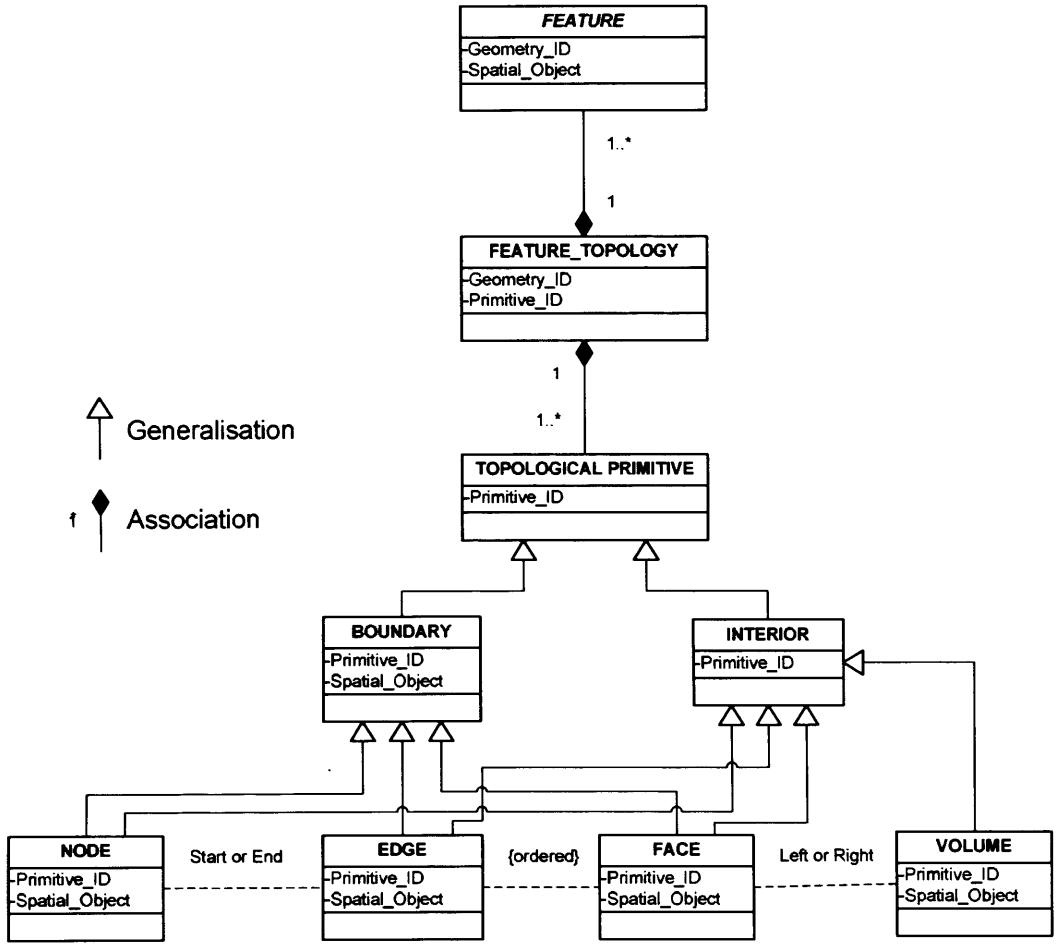


Figure 29 - Model of Topology - Showing Link to Geometry

STS is optimised for the efficient querying of binary topological relationships between objects and has been designed to allow a wide range of object types to be represented. It does not enforce relationships between the primitives – if required this validation is performed by the topological engine. This approach allows STS to take advantage of available representations of the 3D primitives available in different Object-Relational databases – STS stores the spatial object representing the each primitive along side the primitive itself, rather than building the spatial representation of Faces, Edges and Volumes from coordinates stored against the Node primitive by following joins from Node to Edge to Face and Volume. In STS, therefore, coordinate information is retrieved from the highest dimension primitive (assuming that this can be handled by the database) – to reconstruct a Surface, coordinate information is retrieved from the Faces, rather than from the Node primitives as would be the case in the original version of 3DFDS (see Section 5.10 for further discussion of this approach).

Similarly, in Figure 29 the presence of spatial objects associated with the Node, Edge, Face and Volume primitives is optional rather than mandatory. Their implementation will depend on the selected database. Again, this approach has been taken in order not to limit the scope of the

structure. For example, if the selected database does not support curved Surfaces, then the spatial object representing the Face primitive does not need to be populated, provided that the topological engine can determine the existence of the Face from the parent object.

Formal definitions of the primitives are presented here, with each definition split into a definition the primitive itself, and an optional spatial representation of the primitive.

5.2.2.1 Nodes

A Node is defined as a 0-dimensional primitive. Unlike a Point, it cannot be co-located with another Node. A Node cannot be located within an Edge. If a Node is required within an Edge – for example to model a link to another Edge, then the Edge is split and two Edges are created. However, a Node may be located within the interior of a Face or of a Volume. These definitions are not mutually exclusive. A Node can be identified as the end-Point of an Edge and also be internal to a Volume. Each Node can be connected directly to another Node by at most one Edge.

Node_Geometry

If a spatial object corresponding to the Node is to be created, it is represented by a single tuple of coordinates that define its location.

Relating the Node to the Object

A Node can be used to represent one or more Point objects. Additionally, as Node represents the end-Points of Edges then Nodes are also associated directly with Line, Surface and Body objects. Direct associations are created between the Node primitives and the Line, Surface or Body objects of which they form part. One Node can be associated with many objects.

5.2.2.2 Edges

An Edge is defined as a 1-dimensional primitive. The end-Points of each Edge (if they exist) are defined by Nodes – if Nodes are created to represent the boundary of the Edge, a minimum of two Nodes are associated with each Edge. However, an Edge may have multiple end Nodes. For example, in Figure 30 below Edge 1 and Edge 6 have four end Points. Edges, in turn, are used to define the boundary of Face objects.

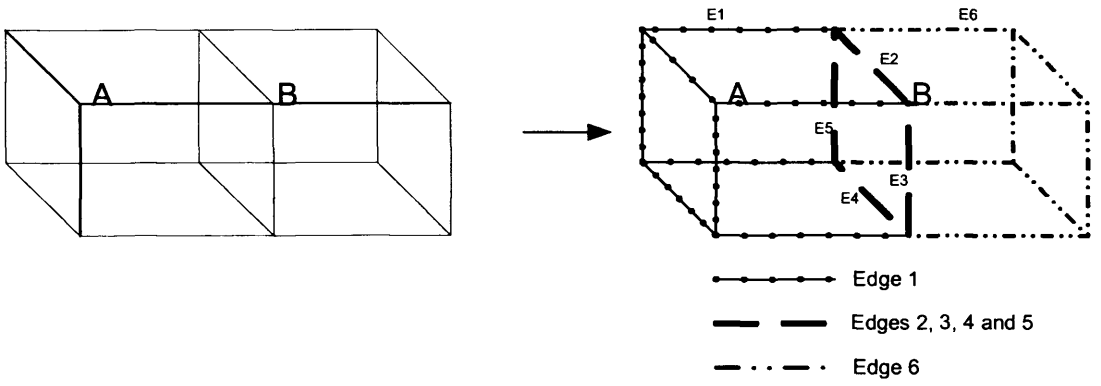


Figure 30 - Edges with Multiple End Points

An Edge may also be located within the interior of a Face or of a Volume. These definitions are not mutually exclusive. For example, an Edge can be included in the boundary of a Face and also be internal to a Volume. An Edge may not intersect with or overlap itself or any other Edge. An Edge cannot pass through a Node or through a Face without being broken into separate Edges. Two or more Edges may only meet at a Node. An Edge may be completely contained within a Face or a Volume. An Edge may not be partially contained in a Face or Volume.

Edge_Geometry

If a spatial object corresponding to the Edge is to be created, the implementation will depend on the spatial object type within the selected database environment. An Edge can be defined by an ordered list of 3D coordinate tuples. Edges may be curved or straight, and may be formed of one or more segments, which may also be curved or straight. Additionally, if the spatial object type supports the definition of an Edge as a parametric equation, then this Edge is also considered valid within the context of STS. Similarly, more complex definitions for Edges with multiple end Points can also be implemented provided that these are supported by the underlying spatial object. This is possible as Edge geometries are held as spatial objects with the Edge primitives themselves, rather than being constructed from the coordinate information held on the Nodes. In the formal definition of STS, any intermediate 3D coordinate tuples used to define the geometry of an Edge are NOT considered to be Nodes.

Relating the Edge to the Object

An Edge can be used to represent the interior of one or more isolated Line objects, with the associated end-Nodes representing the boundary of the Line. Additionally, as Edges bound Faces which in turn bound Volumes, Edges can also be used to represent the boundary of Surface and Body objects. Direct associations can be created between the Edge primitives and

each of the three object types that an Edge can represent. One Edge can form part of many objects.

5.2.2.3 Faces

A Face is defined as a 2 or 2.5-dimensional primitive. The boundary of the Face, if it exists, is defined by one or more Edge objects. Additional Node or Edge objects may also be included in the interior of the Face (i.e. form holes in the Face). Faces in turn define the boundary of Volume objects. A Face may also be located within the interior of a Volume. These definitions are not mutually exclusive. For example, a Face can be included in the boundary of one Volume and also be internal to another Volume.

All bounding Nodes and Edges are considered part of the Face. Any interior Nodes, Edges or Faces, also form part of the Face. A Face may not overlap itself or any other Face. Two Faces may meet only at common Nodes (interior or exterior) or along one or more common Edges (interior or exterior).

Face_Geometry

The geometry of a Face is defined by a collection of three or more closed, ordered coordinate tuples. Alternatively, if supported by the spatial object in the selected database, the Face may be defined through a series of equations or as a curved Surface. Faces are not necessarily planar, and may be composed of multiple planar components as shown in Figure 31, where Body A is decomposed into two Face primitives (F1 and F3) and Body B is decomposed into two Face primitives (F2 and F3). Again, these options are available as the Face geometry in STS is not constructed through relationships with Edges and hence with the Nodes, but is held on the Face object itself.

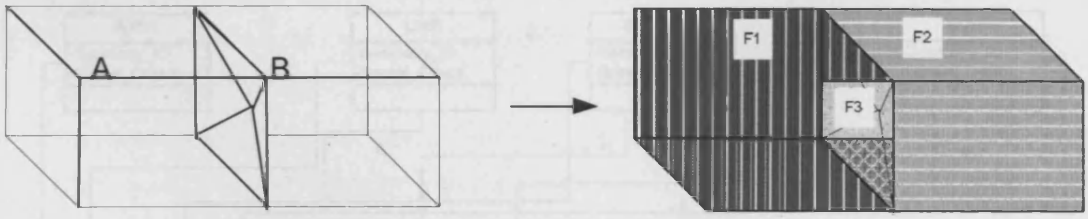


Figure 31 - Multi-Planar Faces

Faces may contain holes, which are defined by one or more (ordered) coordinate tuples contained within the Face. A Face has two sides (left or right). The right side of the Face is defined as the side for which the external boundary coordinates are listed clockwise around the Face, with the left having coordinates in counter-clockwise order.

Relating the Face to the Geometry Objects

A Face can be used to represent the interior of one or more isolated Surface objects, with any Edges and Nodes associated with the Surface representing the boundary of the Surface unless they represent containment exceptions. Additionally, as a Face represents any 2 or 2.5-dimensional object part, then Faces also form the boundary of Body objects. One Face may be associated with many Body objects.

5.2.2.4 *Volumes*

A Volume is a 3D geometric primitive that is composed of the closed region of space bounded by a collection of one or more Faces. A Volume may not intersect with or overlap itself or any other Volume. Volumes may contain cavities, each of which is defined by an inner boundary consisting of a collection of one or more Faces. They may also contain tunnels. Additionally, isolated Nodes, Edges and Faces may be contained within the Volume object.

Volume_Geometry

Depending on the selected implementation, the Volume geometry may be defined in a number of ways – for example by a combination of Faces. Volume geometries can be held against the Volume primitive object if this option is available in the database, rather than being constructed from Nodes via Edges and Faces.

Relating the Volume to Body Objects

A Volume object represents the interior of a Body object, with any Faces, Edges and Nodes also associated with the Body representing the boundary of the Body. One Volume may be related to many Body objects.

5.2.3 Part III - Linking the Geometry and Topology

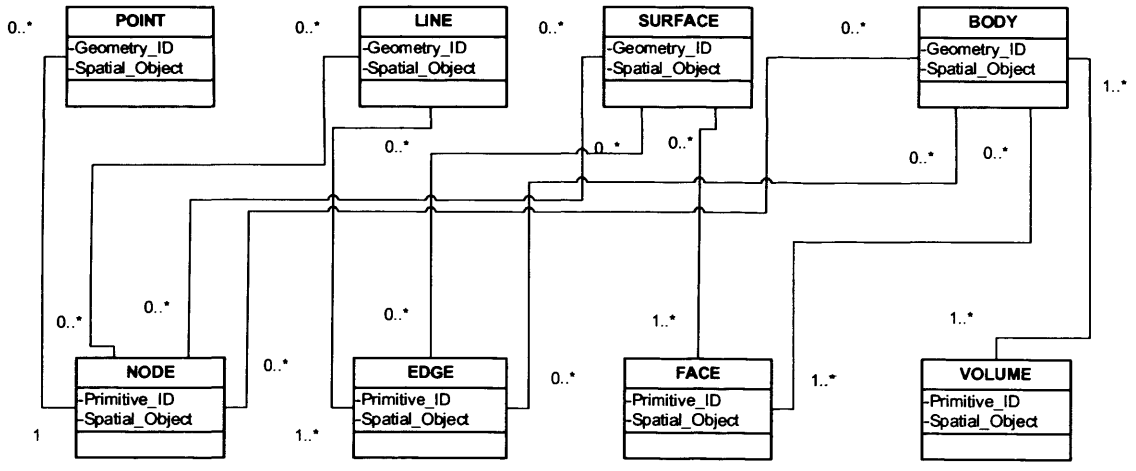


Figure 32 - Linking Topology and Geometry

Figure 32 summarises the relationships between the objects and their topological primitives. In STS Node primitives can be directly associated with Point, Line, Surface and Body object types. Nodes can represent the interior of Point objects. They can also represent part of the interior (as containment exceptions) or boundary of Line objects, Surface objects and Body objects. Similarly, Edges can represent the interior of Line objects, and can also represent the interior (if they are contained) or boundary of Surface or Body objects. Face primitives can represent the interior of Surface objects, and the boundary or interior of Body objects.

STS is designed to optimise query performance for 9-Intersection relationship queries in a 3D setting. Additionally, in order to support 3D objects having the characteristics described in Chapter 4, flexibility to handle a wide variety of 3D objects was also considered as part of the design process. The 9-Intersection relationships focus on the interior, boundary and exterior of objects. Thus, provided that the topological engine can determine the interior and boundary of an object, and model these in terms of Node, Edge, Face and Volume primitives, then the object can be incorporated into STS and topological relationships involving this object identified. To this end, STS does not enforce the Node/Edge/Face/Volume hierarchy, although this can be enforced by the topological engine – in fact, the following rules apply for STS primitives (Table 26).

Object Dimension	*Interior Primitive (Must Have)*	*Boundary Primitive (Must Have)*	*Boundary Primitive (May Have)*	*Boundary Primitive (May Have)*
3D	Volume(s)	Face(s)	Edge(s)	Node(s)
2D	Face(s)	Edge(s)	Node(s)	N/A
1D	Edge(s)	Node(s)	N/A	N/A
0D	Node(s)	N/A	N/A	N/A

Table 26 - Object/Primitive Combinations in STS

Exceptions to these rules do, however, exist. A closed 1D Line does not have a boundary, and therefore may not have any associated Node primitives unless it intersects with other objects or primitives are created arbitrarily by the topological engine. This is shown in Figure 33 below, where (a) represents an isolated closed Line object, having no end Points and hence no defined boundary Node. Figure 33(b) represents the insertion of a Node on this Line, created due to the intersection of the Line and the adjacent square. In this case, the Node represents an interior primitive of the Line. Similar rules apply for a closed 2D Surface (such as a sphere) embedded in 3D space.

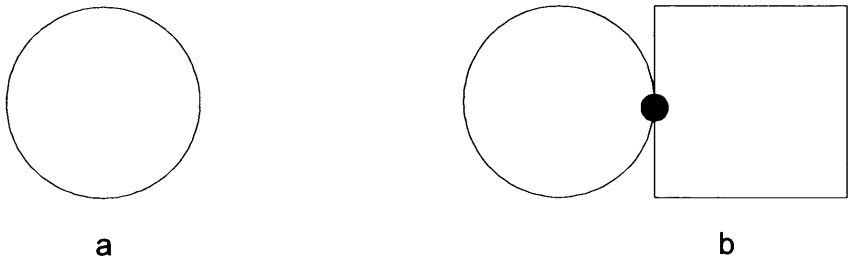


Figure 33 - Absence of a Boundary Node for Closed Lines

In STS, ALL primitives forming part of an object are directly associated with that object. This approach contrasts with that taken by traditional B-Rep structures including 3DFDS, where, for example, Surface objects are only associated directly with Face primitives. Associated Arc and Node primitives (i.e. the representations of the boundary of the Surface) are then identified by following joins through the Face object. In STS, direct queries of the Surface/Node and Surface/Edge relationships are used to identify these boundary primitives.

5.3 Logical Model

A logical model consists of the translation of the conceptual schema defined above into the data model supported by the selected database management, taking into account, for example, the types of spatial objects available within the database. Thus the Database Management System (DBMS) used in the implementation must be one that supports the logical model, although the logical model can be applied to many DBMS. STS is described in the context of a DBMS supporting spatial object types.

The logical model for STS is presented in Figure 34 below. The object types of Point, Line, Surface and Body are combined into a single object called FEATURE, which can also represent the complex elements of these objects, including cavities, tunnels and holes. Multi-part (compound) objects are split into their simple components through TOPO_PART_TABLE, which breaks the spatial object into one or more simple object parts (although these simple objects are not represented by spatial objects).

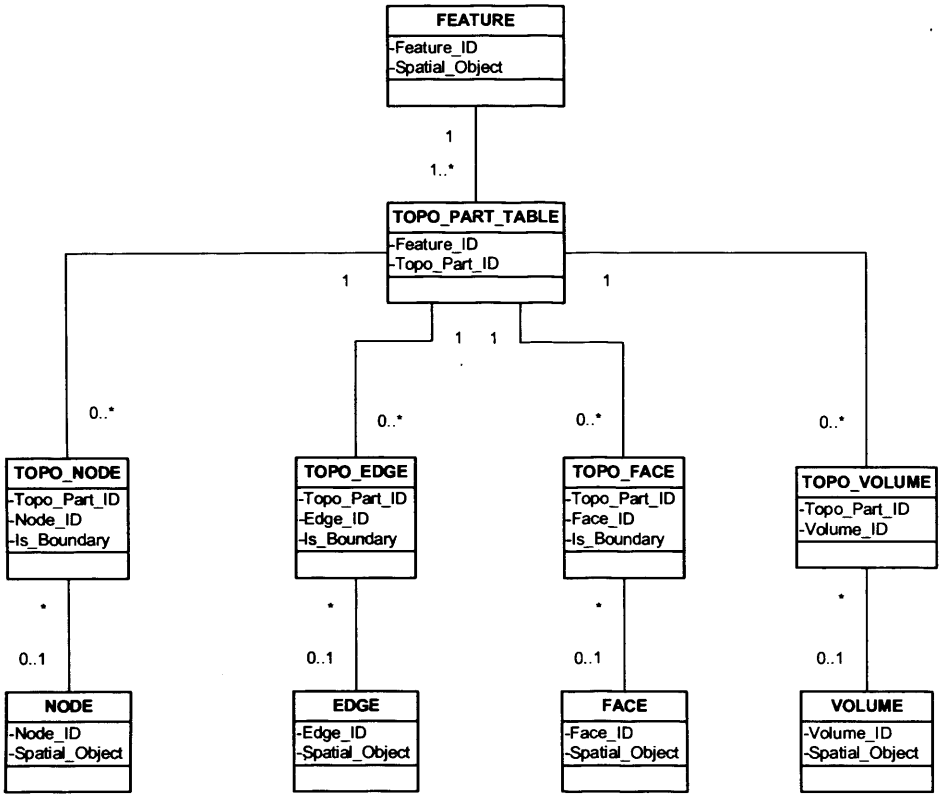


Figure 34 - Simplified Topological Structure

Tables (TOPO_NODE, TOPO_EDGE, TOPO_FACE and TOPO_VOLUME) have been created in the logical model to implement the many:many relationship between the primitives and the object types. An IS_BOUNDARY flag has been added to the join to define the relationship further – is the associated primitive a boundary primitive or interior to the object in question. The spatial objects representing the primitives are stored in separate tables, as they are not required for binary relationship identification. STS does not mandate the representation of primitives as spatial objects, although this is likely to occur in practice as the objects may be utilised in the structure population and maintenance processes. Although a spatial object representing a3D Volume is not currently supported by Oracle 10g (Oracle 2006a), it is included in the above diagram for completeness (and will be available in Oracle 11g, Oracle 2007).

5.4 Validating the STS

STS has been developed to incorporate topological functionality into 3D GIS and to implement the mapped 9-Intersection relationships within this environment. The structure is reviewed from the perspective of:

- The support provided for situations encountered with real-world 3D data.
- The support provided for the 9-Intersection relationships identified by Zlatanova (2000).

Note that the Figures in this Section repeat those shown in Section 4.9 (Validating the 3DFDS Approach) to facilitate comparison between the structures.

5.4.1 Handling 3D Objects

5.4.1.1 *Complex Objects*

Complex objects, i.e. those containing cavities, tunnels or holes, are supported by STS as shown below. The Faces of internal cavities are modelled as BOUNDARY primitives of the object.

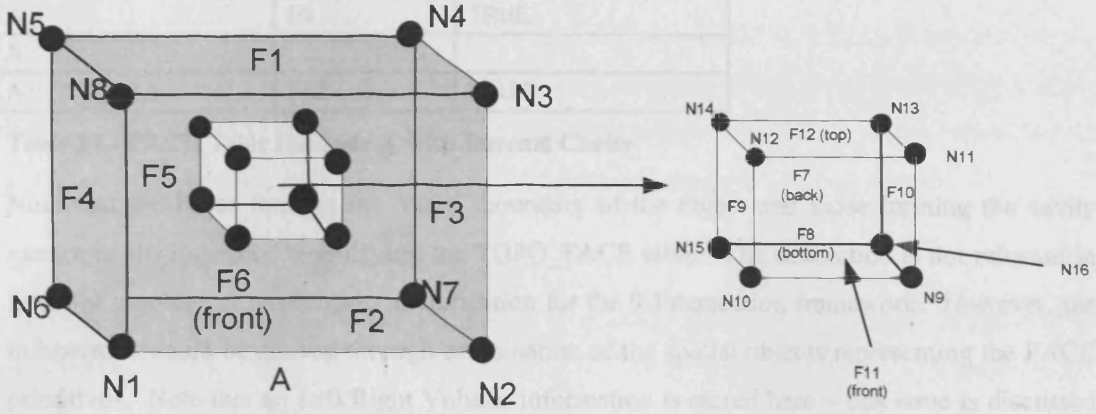


Figure 35 - Body with Internal Cavity, showing Topological Primitives

Thus the TOPO_FACE table for Body A in Figure 35 will contain the following entries (Table 27).

TOPO_PART_ID	*FACE_ID*	*IS_BOUNDARY*
A	F1	TRUE
A	F2	TRUE
A	....	
A	F6	TRUE
A	F7	TRUE
A	F8	TRUE
A	F9	TRUE
A	....	
A	F12	TRUE

Table 27 - FACE Table for Body A with Internal Cavity

Note that the Faces forming the 'outer' boundary of the object and those forming the cavity cannot be distinguished by querying the TOPO_FACE table. The distinction is not relevant in terms of topological relationship identification for the 9-Intersection framework. However, the information could be derived through examination of the spatial objects representing the FACE primitives. Note that no Left/Right Volume information is stored here – this issue is discussed further in Section 5.6.

5.4.1.2 Compound Objects

Compound objects are supported in STS through the use of a TOPO_PART_TABLE, which allows objects to be split into simpler constituent objects. This in turn allows the determination of the topological relationship between parts objects, as well as of the objects as a whole. Note that this approach differs from that taken for objects having cavities – for a single Body having a cavity, only one record in the TOPO_PART_TABLE is created, and then linked to both the Faces forming the outer shell and all the Faces forming the inner shell (with all Faces having IS_BOUNDARY = TRUE, see .

5.4.1.3 Overlapping Objects

STS supports overlapping objects by means of the many:many relationships between primitives and object parts in the TOPO_PART_TABLE. These relationships, represented by the four TOPO_ tables, allow one Node to represent many Points, one Edge to represent many Lines and so forth. Splitting objects into multiple parts may be required, as no overlapping primitives are permitted (although the primitives themselves do not necessarily provide a full partitioning of the space, unlike those described by Penninga *et al.* 2006 – in other words, STS does not explicitly represent the air between buildings).

5.4.1.4 *Curved Surfaces*

Support for curved Surfaces in STS is dependent on available representations in the implementation environment. As with 3DFDS, curved Surfaces can be decomposed into triangles or other planar Surfaces. However, as there is no enforced relationship between primitive types in STS, other representations are possible. Additionally, the representation of these primitives as spatial objects is optional within the STS schema.

5.4.1.5 *Non-Manifold Objects*

As with curved Surfaces, representation of non-manifold objects in STS is dependent on the implementation environment. In this case, provided that the engine can distinguish the interior and boundary of such objects, and represent these as entries in the TOPO_ tables, non-manifold objects can be supported. Again, spatial representation of the primitives is optional.

5.4.2 Support for 9-Intersection Relationships

The first part of the relationship determination process involves the identification of the interior and the boundary of the object. In the context of B-Rep the interior of the object comprises the primitives having the same dimension as the object itself. Boundary primitives are those having dimension less than that of the object itself. As shown in Table 26 above, STS mandates primitives having dimension equal to the object in question – these represent the interior of the object. Additionally, primitives having dimension of one less than the object are required (providing the object is not a closed Line or similar). These represent the boundary of the object. Additional primitives are included in STS only when required due to object intersection.

5.4.2.1 *Example 1 – Adjacent Bodies – R287*

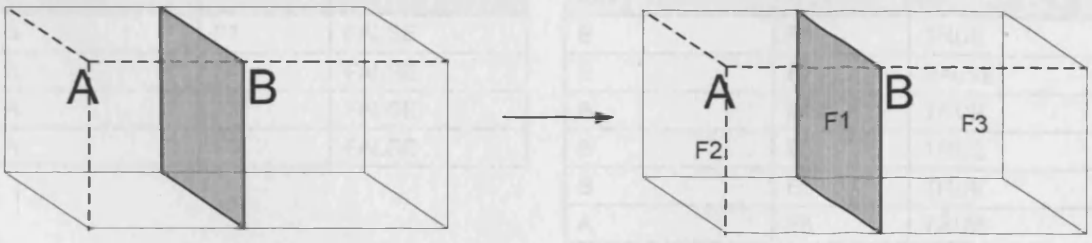


Figure 36 - Body/Body Adjacency - R287

An isolated 3D Body in STS can be represented as one Volume primitive (the interior) and one associated closed Face primitive (the boundary). However, as shown in Figure 36 above, if the Body is adjacent to a second object, a shared Face (F1) is required to represent this relationship. Table 28 shows the TOPO_FACE and TOPO_VOLUME entries for A and B above.

TOPO_PART_ID	FACE_ID	IS_BOUNDARY
A	F1	TRUE
A	F2	TRUE
B	F2	TRUE
B	F3	TRUE

TOPO_PART_ID	VOLUME_ID
A	V1
B	V2

Table 28 – (a) TOPO_FACE and (b) TOPO_VOLUME for R287

5.4.2.2 *Example 2 – Surface/Surface Overlap – R511*

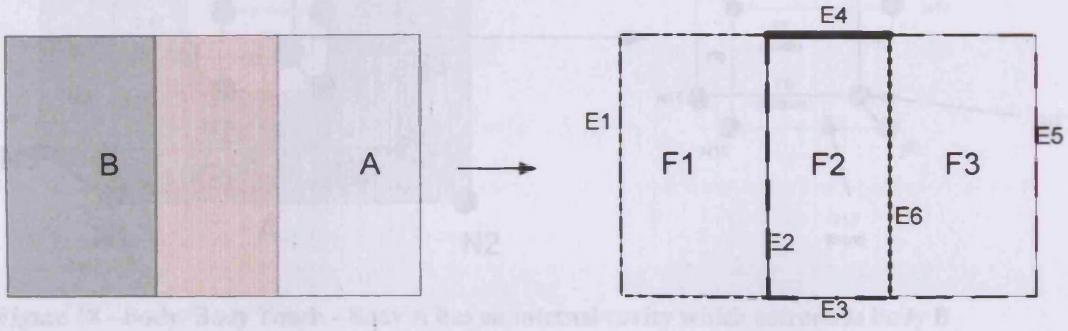


Figure 37 - Surface/Surface Overlap - R511

Table 29a and b below show entries for the Surface/Surface overlap relationship R511. An individual Surface in STS can be represented as a single Face primitive, with an Edge primitive to define its boundary (unless the Surface is closed). However, Surface intersection as shown in Figure 37 results in the creation of additional Edge and Face primitives to represent the shared primitives between the Surfaces. The TOPO_EDGE and TOPO_FACE tables for R511 are shown in Table 29 below.

TOPO_PART_ID	FACE_ID	IS_BOUNDARY
B	F1	FALSE
B	F2	FALSE
A	F2	FALSE
A	F3	FALSE

TOPO_PART_ID	EDGE_ID	IS_BOUNDARY
B	E1	TRUE
B	E2	FALSE
B	E4	TRUE
B	E3	TRUE
B	E6	TRUE
A	E5	TRUE
A	E6	FALSE
A	E3	TRUE
A	E4	TRUE
A	E2	TRUE

Table 29 – (a) TOPO_EDGE and (b) TOPO_FACE for R511

Edge E6 is associated with both A and B. However, this is a boundary Edge for B but an interior primitive for A – thus the $\partial A \cap B^0$ relationship is TRUE. The opposite situation is true for Edge E2, which is an interior primitive for B and a boundary primitive for A.

5.4.2.3 *Example 3 - Body/Body Touch – With Cavity – R287*

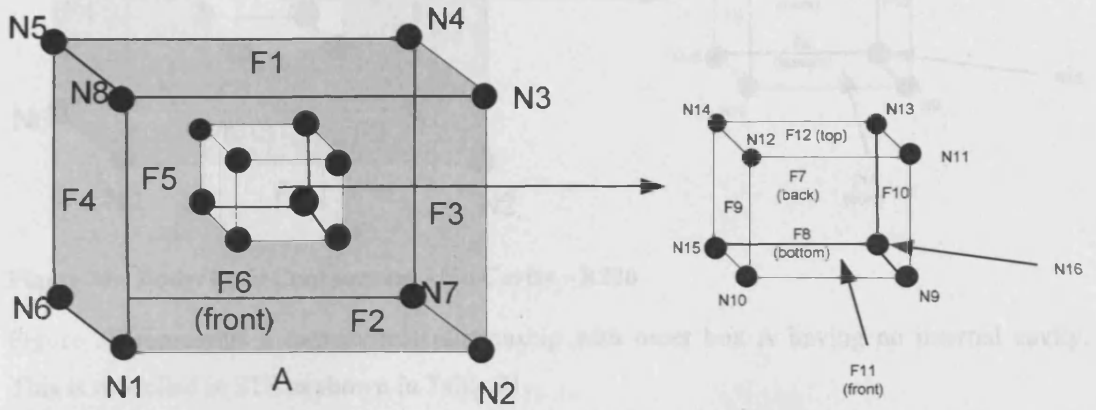


Figure 38 - Body/Body Touch - Body A has an internal cavity which surrounds Body B

The cavity is a boundary of the containing box. Assuming that A is the outer box and B is the inner box (i.e. enclosed by the cavity), Table 30 shows the resulting TOPO_VOLUME and TOPO_FACE entries in STS.

TOPO_PART_ID	*VOLUME_ID*
A	V1
B	V2

TOPO_PART_ID	*FACE ID*	*IS_BOUNDARY*
A	F1	TRUE
A	F2	TRUE
A	...	TRUE
A	F6	TRUE
A	F7	TRUE
A	F8	TRUE
A	..	TRUE
A	F12	TRUE
B	F7	TRUE
B	...	TRUE
B	F12	TRUE

Table 30(a) and (b). TOPO_VOLUME and TOPO_FACE tables for R287

In this case, Ext(A) ∩ Bnd(B) will return FALSE as the Faces making up the boundary of B are ALL shared with A. The shared Faces are marked as boundary Faces for A giving Int(A) ∩ Bnd(B) AS FALSE. Bnd(A) ∩ Bnd(B) will return TRUE. Note that A does not reference the Volume object describing the interior of B. Note that the above description has made an assumption that each Face is not multi-planar, to facilitate comparison with 3DFDS – however this is not enforced in STS (see Section 5.2.2.3).

5.4.2.4 *Example 4 - Body/Body Containment – No Cavity - R220*

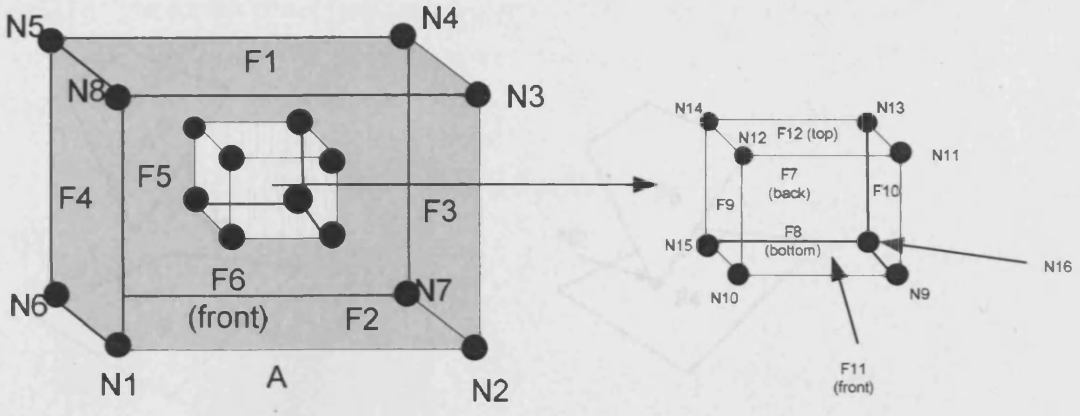


Figure 39 - Body/Body Containment - No Cavity – R220

Figure 39 represents a containment relationship with outer box A having no internal cavity. This is modelled in STS as shown in Table 31.

TOPO_PART_ID	*VOLUME_ID*
A	V1
B	V2
A	V2

TOPO_PART_ID	*FACE ID*	*IS_BOUNDARY*
A	F1	TRUE
A	F2	TRUE
A	...	TRUE
A	F6	TRUE
A	F7	FALSE
A	F8	FALSE
A	..	FALSE
A	F12	FALSE
B	F7	TRUE
B	...	TRUE
B	F12	TRUE

Table 31(a) and (b) TOPO_VOLUME and TOPO_FACE tables for R220

At first glance, the primitives created to represent the R220 relationship are identical to those shown in Figure 38 above. Again, Ext(A) ∩ Bnd(B) will return TRUE as the Faces making up the boundary of B are shared with A. However Faces F7 – F12 are marked as non-boundary for A but boundary for B, giving Int(A) ∩ Bnd(B) as TRUE. Additionally, due to the shared Volume V2, Int(A) ∩ Int(B) is also TRUE - A references this shared Volume object, and the interior of A is now formed from the sum of V2 and V1.

5.4.2.5 *Example 5 – Surface/Surface Touch – R063*

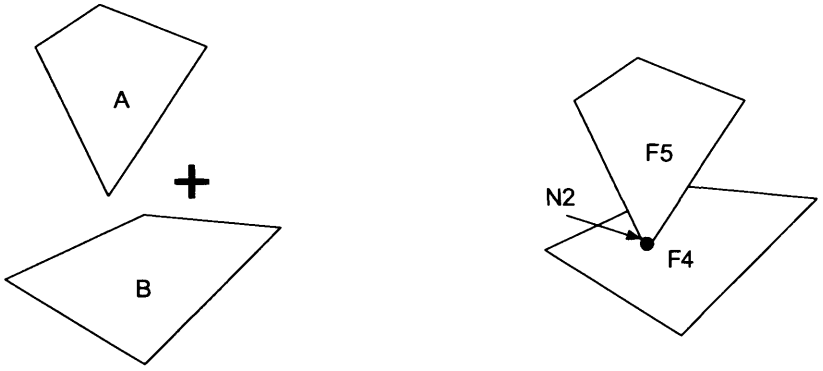


Figure 40 - Surface/Surface Touch - Simple Surfaces

STS handles this situation by associating additional, non-boundary Node N2 with the unsplit Face F4. This results in the following entries in the TOPO_NODE and TOPO_FACE tables.

TOPO_PART_ID	*NODE_ID*	*IS_BOUNDARY*
A	N2	TRUE
B	N2	FALSE

TOPO_PART_ID	*FACE_ID*	*IS_BOUNDARY*
A	F5	FALSE
B	F4	FALSE

Table 32 – (a) TOPO_FACE and (b) TOPO_NODE for R063

5.4.2.6 *Example 6 – Surface/Surface Touch – R287*

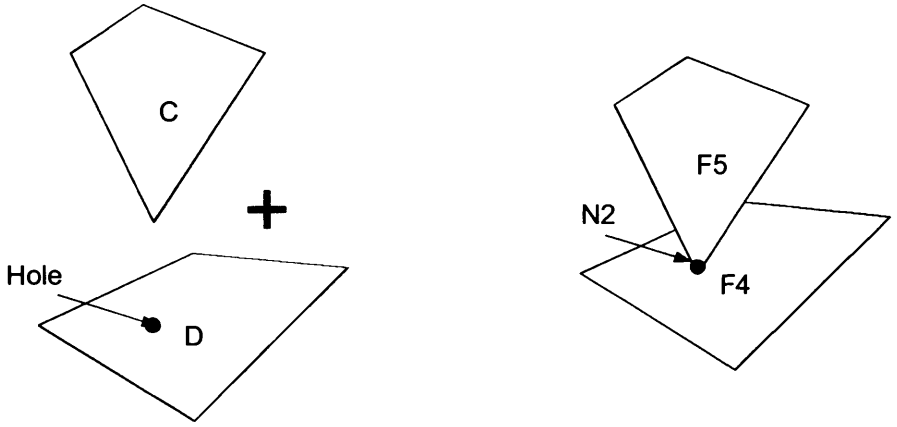


Figure 41 - Surface-Surface Touch – Existing Hole in Surface

TOPO_PART_ID	*NODE_ID*	*IS_BOUNDARY*
C	N2	TRUE
D	N2	TRUE

TOPO_PART_ID	*FACE_ID*	*IS_BOUNDARY*
C	F4	FALSE
D	F5	FALSE

Table 33 – (a) TOPO_NODE and (b) TOPO_FACE for R287

The STS representation of the relationship shown in Figure 41 (R287) is shown in Table 33. Node N2 is a BOUNDARY Node in F4 (i.e. is a hole in F4 as it already exists in the Face

before intersection is computed), and should thus be included as part of the Surface should Surface D be reconstructed from its primitives. This contrasts with the situation illustrated in Table 32 above, where N2 is not a boundary Node, and is only created due to the intersection of the two Faces.

5.4.2.7 *Example 7 – Body/Line Containment – R220*

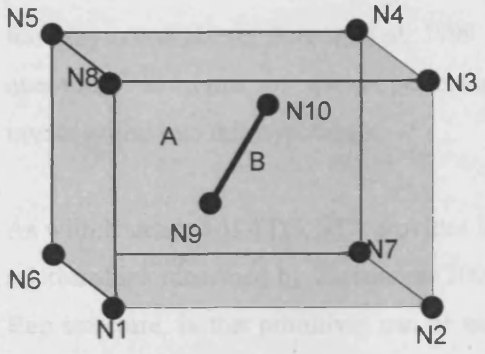


Figure 42 - Line B contained within Body A

A Line-in-Body relationship (R220) is shown in Figure 42 where the Body is Object A and the Line is Object B. Assuming that the Nodes forming Body A have ID values N1 to N8, and the Nodes for Object B have ID values N9 and N10, Edges for A are numbered E1 to E12 and the Edge associated with B is E13, the TOPO_NODE and TOPO_EDGE tables will be populated as follows:

TOPO_ PART_ID	*NODE ID*	*IS_BOUNDARY*
A	N1	TRUE
A	N2	TRUE
A	...	TRUE
A	N8	TRUE
A	N9	FALSE
A	N10	FALSE
B	N9	TRUE
B	N10	TRUE

TOPO_ PART_ID	*EDGE ID*	*IS_BOUNDARY*
A	E1	TRUE
A	E2	TRUE
A	....	TRUE
A	E12	TRUE
A	E13	FALSE
B	E13	FALSE

Table 34 (a) and (b) TOPO_NODE and TOPO_EDGE, Line in Body relationship

Note that single-segment Edges have been used in this example for clarity and to facilitate comparison with 3DFDS. However, in STS it is also possible to represent Body A about as a closed volume having no Edges and a single Face (assuming that it does not intersect any other objects).

5.5 *Strengths of STS*

STS presents a more compact structure than 3DFDS both in terms of the number of tables queried and in terms of the storage required (see Section 6.4.2). In fact, STS limits the number of tables to be queried for 9-Intersection relationship determination to four, in contrast to the nine for basic 3DFDS, and the 15 for Extended 3DFDS. This reduces the number of relational joins to be followed for each query. It is hypothesised that this will reduce query performance time (as noted above, Atzeni *et al.* 1999 note that relational joins are one of the more costly operations in terms of query performance). Chapters 8 and 9 describe experimental investigation into this hypothesis.

As with Extended 3DFDS, STS provides full support for the determination of the 9-Intersection relationships identified by Zlatanova (2000) and also incorporates the general benefits of a B-Rep structure, in that primitives can be used to reconstruct objects and support is provided for other frameworks utilising concepts of interior, boundary and exterior or combinations of these. The structure presents a systematic, simplified, standardised approach to containment exception handling at each dimension, rather than making use of specific exception tables. Using this approach STS can also be extended to dimensions higher than 3.

Although based on B-Rep, STS relaxes the rules enforcing a hierarchical relationship between the Node, Edge, Face and Volume primitives. It is assumed that, if required, these rules can be enforced by the topological engine, allowing STS to represent a wider range of objects than traditional B-Rep structures, and to take full advantage of the spatial object types supported within Object-Relational environments, such as curved Lines or Surfaces. Spatial representation of primitives is not mandated, although this may be required to support the process of structure population and maintenance.

The containment exception records in the TOPO_ tables do require additional data storage, as does the direct association of all primitives with the object. Both of these will result in larger TOPO_ tables and hence slower query performance. This can be mitigated by use of a combined index on the TOPO_PART_ID and IS_BOUNDARY fields. Additionally, that STS only requires two levels of primitives to be created:

- Those having dimension identical to the object, representing the interior of the object.
- Those having dimension one less than the object, representing the boundary of the object (if this exists).

Further levels of primitives are only required where they are shared between multiple objects.

No join tables exist between the primitives, reducing storage. STS also supports relationship determination in the context of the other Frameworks identified by the ISO 19107 standard (OGC 2006) – i.e. the Set Theoretic and Full operators.

5.6 Limitations of STS

Hoffman (1989) states that one advantage of a B-Rep structure is that "[it] can represent a solid unambiguously by describing its Surface and topologically orientating it such that we can tell, at each Surface Point, on which side the solid interior lies." Orientation is therefore important to identify the inside and outside of a 3D solid. In the case of STS the determination of this orientation must be carried out by the topological engine, as it is no longer stored in the data structure itself (see Section 5.10). The lack of an explicit hierarchy between the primitives may also increase engine algorithm complexity. For example, although STS provides a direct list of the Faces making up a Body, it does not identify which of these Faces represent the outer boundary of the Body and which (if any) Faces represent internal cavities. In the case of a Node on Face containment situation where the parent object has multiple Faces, it is not possible to determine which Face contains the Node in question.

The lack of Left and Right Body information also complicates the process of identifying the interior and exterior of a Volume primitive – of particular relevance in containment situations. It is proposed here that this issue can be overcome within the topological engine, as the coordinates making up each Face are available, as is a list of Faces making up a Volume. Should performance prove inadequate, it is also possible to extend STS to model this relationship directly – an additional field can be added to the TOPO_FACE table for this purpose, with the orientation of the FACE taken from the order of coordinates stored in the spatial representation of the primitive. This approach allows STS to model the situation described in Figure 39, where more than one Volume is on the same side of a Face.

STS is designed to support frameworks based on the determination of the interior, boundary and exterior of objects and also provides the ability to determine the dimension and number of any shared primitives. Any topological frameworks not based on either one or both of these concepts are thus not supported by the structure.

5.7 Valid Objects in STS

The above discussion has taken place in the context of applying STS to model topological relationships between typical data types utilised in 3D GIS – Points, Lines, Areas/Surfaces and Bodies. Unlike 3DFDS, STS does not enforce links between the Node, Edge, Face and Volume

primitives. This removes automatic validation provided by the constraints within the structure that a 3D object stored in STS is manifold[10].

The validation checks that can be enforced by relational constraints in the original 3DFDS (as described by Rikkers *et al.* 1994) include:

- Does each Edge link to exactly two Nodes
- Does each Face link to only two Bodies

Note that in 3D, the number of Faces linked to an Edge cannot be enforced by the structure. It is also not possible for the structure to validate that a Face is made up of at least three coordinate points.

Although at first the lack of direct relational joins between the primitives appears to disadvantage STS for use in applications where manifold objects are required, in practice structure validity would in fact be tested in the Topological Engine implementing rules including the Euler Equation, validating planarity of a Surface/Area, validating closure of an object, validating the number of coordinate points making up a Face, validating that Faces do not intersect or self-intersect and so forth (see Appendix 1 for examples of such algorithms). These tests would be applied before the database is populated, rather than being validated by the constraints in the database. In both 3DFDS and STS, the use of triggers on the database tables will ensure that should a user add, edit or delete a topological primitive, the engine update and validation routine would be executed to maintain data consistency and validity.

The lack of enforced links in STS offers advantages in terms of the range of objects that can be stored, extending this beyond the capability of 3DFDS. For example, a body which does not intersect with any other object can be modelled as a single Volume primitive and a corresponding single closed Face, thus reducing storage requirements for Edges and Nodes. A closed line can be represented as a single Edge without the need to artificially insert one or more Nodes. STS can also model non-manifold objects (see 5.9.1.1 below).

Furthermore, the relaxation of this hierarchy means that there is no requirement for an object to have any form of spatial representation at all in STS, allowing relationships such as those described in Table 3 in the context of Chemistry to be modelled, provided that the concept of 'interior' and 'boundary' can be mapped to the objects being modelled. Similarly, STS can be used to model the topological links identified between members of ontologies. Experiments

[10] For a discussion on the impact of the other consequence of removing these links – the inability to reconstruct primitives from those of lower dimension – see Section 5.10.

conducted by Agarwal (2005) link the concepts of *place* and *neighbourhood*, by *intersects* and *containment* relationships.

Van Oosterom *et al.* (2006) detail relationships between temporal points (before, equal, after) and five topological relationships between temporal intervals (disjoint, touch, overlap, included, equal). STS can be utilised to model these relationships. Assume that an Edge primitive can be used to represent a time interval. The overlap relationship between two time intervals A and B (Figure 43) can be represented as shown in the TOPO_NODE and TOPO_EDGE tables in Table 35, without requiring spatial representation of the primitives. As the overlap relationship is defined by means of a shared primitive having the same dimension as the original object, it would also be possible to represent this relationship in terms of Edges, without associated Node primitives. The selection of the Edge primitive to represent time intervals is also arbitrary – Face or Volume primitives could also be selected, utilising equal storage space and giving identical query performance.

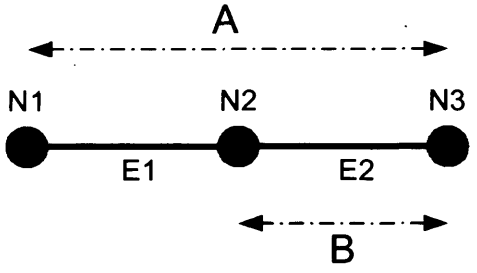


Figure 43 - Overlapping Time Lines

TOPO_PART_ID	*NODE_ID*	*IS_BOUNDARY*
A	N1	TRUE
A	N2	FALSE
A	N3	TRUE
B	N2	TRUE
B	N3	TRUE

TOPO_PART_ID	*EDGE_ID*	*IS_BOUNDARY*
A	E1	FALSE
A	E2	FALSE
B	E2	FALSE

Table 35 – Overlapping Time Lines

This approach allows relationships beyond the five identified by van Oosterom *et al.* (2006) to be modelled, opening up the potential to describe the relationships between time intervals in terms of the 9-Intersection relationships should this be required (see also Section 11.6.3, which describes further work to extend this concept and integrate STS with temporal points and intervals).

Finally, the lack of a requirement to spatially represent the primitive geometries implies that STS can be used to handle all the four object types described by Arens *et al.* (2005) – tetrahedral, polyhedra, polyhedra with patches and CAD objects, provided that the Topological

Engine can identify the required primitives. For example, if a shared face is a curved surface CAD object, the Face can be inserted into TOPO_FACE but the spatial representation left empty should the database not support CAD objects.

Thus valid objects in STS are those for which a rule base mapping the relationships between objects to the corresponding STS primitives that yield the correct relationship on query. Once defined, the selected rules-base is implemented in the Topology Engine and executed on structure population or modification. All validation (i.e. that the objects do conform to this rule base) will be carried out in the Engine itself, with different components (or plug-ins) being developed to suit different datasets. A different set of validation rules is implemented within the engine for each different valid object type. These sets of rules can include, but are not limited to, those described in Appendix 1.

This flexible approach to the definition of a valid object allows STS to underpin many different datasets and 3D data types, and also allows new 3D data types to be utilised to represent primitives as these become available within the database environment.

5.8 Comparing STS and the ISO 19107 Topology Specification

ISO 19107 (OGC 2006) provides guidance to implementers of topology, allowing them to select areas of the standard with which they wish to comply.

ISO 19107 presents a series of classes to structure basic topological packages and proposes one root object TP_Object, which is an abstract class that supplies a root type for other topological classes. Subclassed from this are TP_Primitive and TP_Complex classes, where a TP_Complex consists of an organised structure of TP_Primitives and a TP_Primitive must form part of at least one TP_Complex. A brief comparison between the ISO 19107 standard and STS is presented here. Figure 44 presents a topological class diagram.

Figure 44 - Topological Class Diagram (OGC 2006)

5.8.1.1 *Topological Primitives*

TP_Primitive class is the root for all the primitives for each dimension and classes for representation of the relationships between these primitives. Classes in this package are split into Boundary and Primitive classes.

Boundary classes describe the boundary of primitives of higher dimension in terms of primitives of lower dimension. These relationships are enforced, unlike in STS where a boundary primitive, while normally present, is not mandatory. Additionally, boundary in STS can be expressed in terms of the primitives of the next lower dimension, whereas in 19107, all lower dimensions are required. Table 36 gives a brief definition of Boundary class definitions, along with their STS equivalent.

Boundary Class	*ISO 19107*	*STS*
TP_EdgeBoundary	Contains two TP_Node references. The startNode has a positive orientation and the endNode a negative orientation.	No direct link in STS between an Edge and its corresponding Nodes. Information can be derived by examining the coordinates of the spatial object representing the Edge.
TP_FaceBoundary	Consists of a number of TP_Rings, with one of these being distinguished as the exterior boundary. Each ring is orientated so that the boundary of the Face is on its left. Each ring consists of a number of TP_Directed Edges.	No links to corresponding Edges are defined in the context of Faces in STS. This information can be derived through the examination of coordinate information associated with the Face.
TP_SolidBoundary	Defined as shells, with one shell being designated as an exterior shell. Each shell is oriented so that the solid is on the bottom. A TP_Shell in turn consists of a number of TP_Faces, but unlike a TP_Ring, it has no natural sort order.	Spatial object representing the Volume can be defined as a collection of shells. However, no link between the shells and the corresponding Faces exists in STS. This information can be derived through the examination of coordinate information associated with the Volume.

Table 36 - Boundary Classes in TP_Primitive

Primitives are orientable with two orientations possible and each primitive is associated with two directed topological entities. The standard allows for association between objects and their topological primitives, but also allows topological primitives to stand alone. Table 37 gives a brief description of TP_Primitive classes along with their STS equivalents. For each primitive, a TP_DirectedPrimitive association is also present, having a positive and negative direction. This assists in the identification of boundary and co-boundary relationships. For example, if a positive directed TP_Edge is on the boundary of a TP_Face, then the positive directed TP_Face is on the co-boundary of the TP_Edge. In contrast, there is no concept of a directed primitive in STS. Direction is encoded in the object associated with Edge and Face primitives.

Primitive Class	*ISO 19107*	*STS*
TP_Node	Has dimension 0 and an empty (NULL) boundary. Each primitive can be associated with two TP_DirectedNode instances. These represent the orientation of the Node with respect to a specific Edge – a "-" association implies that it is a start Node of the Edge.	Node primitive is equivalent, but no direction association present.
TP_Edge	A 1-dimensional primitive, having the boundary defined as a pair of TP_DirectedNodes. Each TP_Edge can be associated with two TP_DirectedEdges, representing the positive and negative directions along the TP_Edge. These in turn can be used to bound a TP_Face	Edge primitive is the equivalent, but no link exists to the Nodes forming its boundary. Direction of an Edge can be derived through examination of the coordinates of the associated spatial object. No link to the Face primitive bounded by the Edge exists.
TP_Face	Has dimension 2, with the boundary being defined as a set of directed TP_Edges. Is associated with two TP_DirectedFaces, representing the positive and negative sides of the TP_Face (defined by examining the orientation of the corresponding TP_Edges. The orientation of a TP_DirectedFace that bounds a TP_Solid will Point away from the solid.	Face primitive is the equivalent, with no links exist to the Edges forming its boundary. Direction of a Face can be derived through an examination of the coordinates of the associated spatial object. No link to the Volume primitive bounded by the Face exists.
TP_Solid	A 3-dimensional primitive with the boundary being defined as a set of TP_Faces or their negative proxies. These TP_Faces are organized into shells, with an indication of which TP_Faces form the exterior of the TP_Solid and which represent cavities. TP_DirectedSolid gives the association between the solid and a specified TP_Face, with a positive direction indicating that the upward normal for the TP_Face Points outwards.	Volume primitive is the equivalent, with no links to the Faces forming its boundary. Where 3D solid representation is not available, the interior and exterior shells of the Volume can be identified through an examination of the Faces associated with the same part object (using the TOPO_PART_TABLE to identify these Faces).

Table 37 –TP_Primitive Classes with STS Equivalents

5.8.1.2 *Operations on TP_Object*

A number of operations are defined on TP_Object, as shown in Table 38, which also contains a brief description of their equivalent in STS.

Operation	*ISO 19107*	*STS*
Dimension	Depends on class of object. 0 for Nodes, 1 for Edges, 2 for Faces, 3 for Solids	As for ISO 19107
Boundary	Always 1 less than the dimension of the original object.	STS generally adheres to this principle but also allows objects such as closed loops to exist without boundary
coBoundary	Returns the collection of objects that have a particular TP_Object on their boundary.	Operation not formally defined for STS but this information can be derived through the use of SQL queries against the FEATURE, TOPO_PART_TABLE and TOPO_ primitive table.
Interior	Returns all the TP_Objects not forming the boundary of the object.	As for ISO 19107.
Exterior	Returns the finite set of TP_Primitives that are in the maximal TP_Complex but not the interior or boundary of this TP_Object	No equivalent in STS, as no concept of maximal TP_Complex
Closure	Union of the interior and boundary.	As for ISO 19107.
Maximal Complex	Returns the maximal TP_Complex for the primitive. This represents the entire collection of TP_Primitives linked to the primitive and not contained in any larger TP_Complex.	This concept is not defined in STS.

Table 38 – Operations on TP_Object and their STS Equivalent

5.8.1.3 *Topology Complex Package*

"A topological complex consists of collections of topological primitives of all kinds up to the dimension of the complex (equivalent to the concept of a layer). Thus a 2-dimensional complex must contain Faces, Edges and Nodes" (OGC 2006). This contrasts with STS, where the equivalent of a complex may or may not contain primitives of lower dimensions. An example of this is an Edge representing a closed Line loop, which does not contain any primitives of lower dimension than 1.

The rules for creation of a TP_Complex are similar to those applying to STS, and are summarised as:

- If two primitives overlap, then subdivide them, eliminating repetitions until there is no overlap.
- Similarly, if a primitive is not simple, subdivide it where it intersects itself, eliminating repetitions until there is no overlap.
- If a primitive is not a Point, calculate its boundary as a collection of other primitives, using those already in the generating set if possible, and insert them into the complex.

- Repeat the steps above until no new primitives are required.

ISO 19107 (OGC 2006) suggests two types of topological complex. Firstly, it proposes a Planar Complex, which has a geometric realisation that can be embedded in Euclidean 2-space. Note that no 3D complex types are defined in the standard. However, STS provides a 3D 'Volumar' equivalent of the 2D Planar Complex (see Appendix 1 for a description of Volumar enforcement, which fully partitions the space with the primitives). The second type of complex is a graph (which may be a separate complex or may be formed from all the Edges and Nodes in an existing complex). The absence of directed Edges currently limits STS when it comes to creating the Edge-Node graphs that can be created as a subset of a topological complex. However, it may be possible to extend STS to include directed Edges, or this information could be derived as required by the topological or networking engine. Alternatively, the hybrid approach suggested by Ramos (2002) may be considered – with two implemented structures, one to support binary relationship determination and the other to support networking.

The two complex types highlight one final difference between the ISO standard and STS - i.e. to the purpose for which they were created. ISO 19107 aims to cover a wide variety of situations involving topology. STS has been designed specifically to optimise the performance of binary topological queries. In ISO 19107, primitives are important in their own right and their interconnectivity is fundamental. In STS, it is the use of these primitives to support binary relationship querying that is the main focus – i.e. their relationship to the main object geometry and FEATURE is fundamental (although relationships between primitives can be deduced (see Section 5.10). This may allow STS to be utilised for binary relationship determination in situations where the geometry of the FEATURE is not defined or not important.

5.9 Comparing 3DFDS and STS

STS has adopted a number of conventions from basic 3DFDS, specifically (where [Edges] represents the terminology used for STS):

- Convention 6 – "Two arcs [Edges] may not intersect; if they do, they should be replaced by four arcs [Edges] joining at one Node".
- Convention 10 – "Arcs [Edges] can leave or enter a Face only at a Node, so an arc cannot intersect a Face".

Additionally, support for situations violating the basic 3DFDS conventions but included in Enhanced 3DFDS is carried forward to the design of STS. These include "Convention 1 – each elementary object should belong to exactly one class i.e. the class system should be complete and exclusive" and "Convention 2: The elementary objects belonging to one class should be of one geometric type". These are handled by the TOPO_PART_TABLE in both 3DFDS and STS, allowing compound objects to be modelled. Convention 11 ("a Node may represent at most one Point object; an arc may be part of at most one Line object; a Face may be part of at

most one Surface object; a Face may have only one Body at its right hand side; a Face may have only one Body at its left hand side; bodies are disjoint") is also violated by both Enhanced 3DFDS and STS. This is also the case in the 2D Formal Data Structure, with De Hoop *et al.* (1993) proposing a Multi-Valued Vector Map approach in 1993.

However a number of differences can also be identified. These are presented here.

5.9.1.1 *Handling Manifold/Non-Manifold Objects*

The B-Rep structure on which 3DFDS is based assumes that in a 3D object an Edge represents a boundary primitive unless it is included in the exception tables. This does not allow this structure to easily represent non-manifold situations such as that shown in Figure 45 below – where the shared Edge N1/N2 forms part of the boundary of some Faces of A (i.e. is not an exception) but may not be a boundary primitive of the Volume depending on the application domain. Thus although the structure can be modelled in 3DFDS the query results returned when determining binary relationships involving this object may be incorrect.

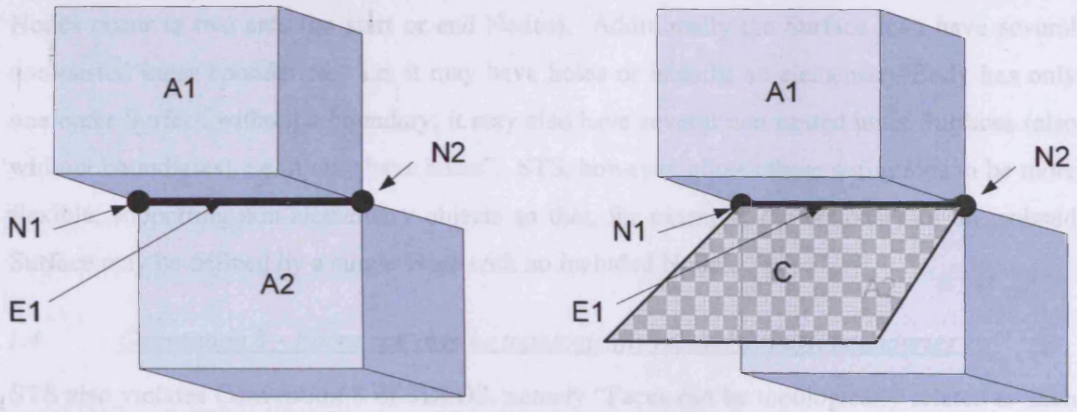


Figure 45 - Non-Manifold Objects

STS on the other hand allows these cases to be decided by the user or the topological engine rather than the data structure – the user can flag the Edge up as boundary or non-boundary as required, once the non-manifold situation is detected by the topological engine populating the structure.

5.9.1.2 *Universe Volume*

Unlike 3DFDS, STS does not implement the concept of a universe Volume to capture situations where one side of a Face does not border a Body object. This is due to the fact that direct relationships between the Face and Volume primitives are not modelled but are instead derived by the topological engine using spatial queries. A universe Volume may be required in STS if this approach proves insufficient for efficient data maintenance (see also Section 5.6).

5.9.1.3 *Conventions 3, 5, 7 and 12*

Convention 3 states that "When a 3D vector map is analyzed as a graph, all Points that are used to describe its geometry are treated as Nodes". The definition of the primitives provided for STS violates this convention – separate Node, Edge or Face primitives are only created by the intersection of two objects. For example, an isolated Body object, which does not interact with any other objects, can be modelled using a single, closed Face to represent its boundary.

Convention 5 of 3DFDS states that "For each pair of Nodes, there is at most one arc [Edge] connecting them directly. The Nodes may also be connected by one or more chains of arcs". STS, however, does not enforce this relationship – more than one Edge can link two Node primitives.

Similarly, Convention 12 states that "an elementary Point object is represented by one Node; an elementary Line object is represented by a simple chain of arcs, with one begin Node and one end Node, no loops and no branches; an elementary Surface object has an outer boundary which consists of only one closed chain of arcs i.e. the arcs occur only once in this chain and the Nodes occur in two arcs (no start or end Nodes). Additionally the Surface may have several non-nested inner boundaries – i.e. it may have holes or islands; an elementary Body has only one outer Surface without a boundary; it may also have several non-nested inner Surfaces (also without boundaries), i.e. it may have holes". STS, however, allows these definitions to be more flexible, supporting non-elementary objects so that, for example, the boundary of an isolated Surface may be defined by a single Edge with no included Nodes.

5.9.1.4 *Convention 8 – Faces can only be topologically related through boundaries*

STS also violates Convention 8 of 3DFDS, namely "Faces can be topologically related to each other only through their boundaries, not through their interiors". 3DFDS requires that the Face is split to handle situations where a surface touches the interior of another, whereas STS handles this situation as a containment case (Figure 40).

5.9.1.5 *Convention 9 – Oriented Edges around a Face*

Convention 9 states that "The (Edges of the) border of a Face should have a unique orientation so that left and right can be defined with respect to the Face". STS does not enforce any relationship between the Edge and Face primitives. However, the orientation may be maintained in the spatial objects representing the Faces and Edges.

5.9.2 Implementing 3DFDS Primitives in STS

In STS the definitions of the primitives and their relationships to simple objects (Point, Line, Surface and Body) have been left as flexible as possible, to support database implementations with different spatial object types. It is possible to apply the definition of primitives used in 3DFDS to STS, without impacting the relationships identified. In particular, Nodes can be created to represent each Point used to describe the objects (Convention 3). Additionally, Faces can be constrained to be planar and bounded by ordered Edges (Convention 7), and the Edges can be constrained to be single segments bounded at each end by a Node (Convention 4). This approach has been taken for the test data creation process, as described in Chapter 6, although it results in additional primitives being created in STS (for example, it is sufficient in STS to represent the intersection of two Surfaces with an Edge, without creating the Node primitives).

Applying 3DFDS constraints to the data creation and maintenance processes may have advantages in terms of facilitating structure population in the case of objects with planar Surfaces. It facilitates conversion of data between the two structures, and also ensures that performance comparisons between the two structures can be made using identical datasets.

5.9.3 Using Object-References

STS has been designed to overcome two limitations of 3DFDS – the number of tables to be queried, and the high number of relational joins required to identify primitives associated with objects. With the advent of Object-Relational databases, relational joins can be replaced by Object-References, which behave like foreign keys (Silberschatz et al. 2002). Oracle (2007c) lists advantages of object references including the fact that they provide an easy mechanism for navigating between objects and the use of dot notation simplifies queries, with Oracle performing the underlying joins. In some cases joins can be avoided, which may improve query performance. Although not considered as part of this research, Object-References may be appropriate for implementation of 3DFDS and of STS, providing potential performance gains in both cases, although this would depend on whether and how the underlying database opts to evaluate the joins.

5.10 Coordinate Retrieval in 3DFDS and STS

Extended 3DFDS and STS have been analyzed in the context of binary topological relationship query performance, and it has been concluded that the information stored in each structure is complete as regards the determination of these in the context of the 9-Intersection Framework. To conclude the description and comparison of the structures, one further consideration can be made – i.e. that of retrieving coordinate information to support the data population and maintenance process. Although the development of a topological engine is beyond the scope of

this thesis, a review of this task may provide a preliminary insight into any required extensions to these structures.

5.10.1 Extracting Coordinate Information in 3DFDS

As 3DFDS is a B-Rep structure, it is possible to reconstruct the geometry higher levels of primitive than Node by examining the relationships between the Node, Edge, Face and Volume primitives. Thus an ordered list of vertices around a Face can be derived by first identifying the ordered list of Edges around the Face and then the Nodes corresponding to those Edges.

A number of modifications to the B-Rep structure have been proposed to improve the efficiency of such queries. These include Wing-Edge representation (Figure 46) which allows single solid polyhedra to be built from component Faces by using the preceding and succeeding clockwise and counter-clockwise Edges, keeping in mind that the Surface normal Points of the Faces point outwards (although this is not valid for a complex). This allows the interior and exterior of an object to be identified (Hoffman 1989, pg. 40).

Figure 46 - Winged-Edge Data Structure (from Hoffman 1989, pg. 40)

Starting from a Node, it is possible to traverse the Edge to the next Node, and then to the next Edge (clockwise or Counter-Clockwise depending on the Face) and so forth, rapidly generating an ordered list of coordinates for a Face. This structure is particularly suitable for procedural implementations due to the iterative nature of the queries, rather than for object-relational database implementation.

A more generic version of the Wing-Edge approach is described by Brisson (1989), who defines a cell-tuple as a combination of cells of all the dimensions, such that each cell (except the full-dimensional one) is in the boundary of the cell of the next dimension in the cell-tuple - in 3D a cell tuple is defined by selecting a volume, one of its faces, one of the edges bounding the face, and a node bounding that edge (Rossignac, 1996). The Switch(k) produces a tuple that has the same elements, except for the element of dimension k. This general operator allows the

identification of the ordered next Edge, Face or Node primitive as required. In this case, the query 'Find the Face Sharing this Edge' requires a Switch(2) operation. 'Find the Next Edge in this Face, sharing this Node' is Switch(1). This approach not only allows a Face to be reconstructed from Node and Edge primitives but also a 3D Volume to be reconstructed from the corresponding Faces. For example, to find the ordered list of Edges around the Face, find one Edge on the Face, then do a Switch(1) to find the next Edge, then a Switch(0) to find the Node at the other end of the Edge, then a Switch(1) again to find the next Edge and so forth. The rationale of such structures is to speed up traversal algorithms for retrieving adjacency and incidence information, although additional storage is required (Silva and Gomes 2005).

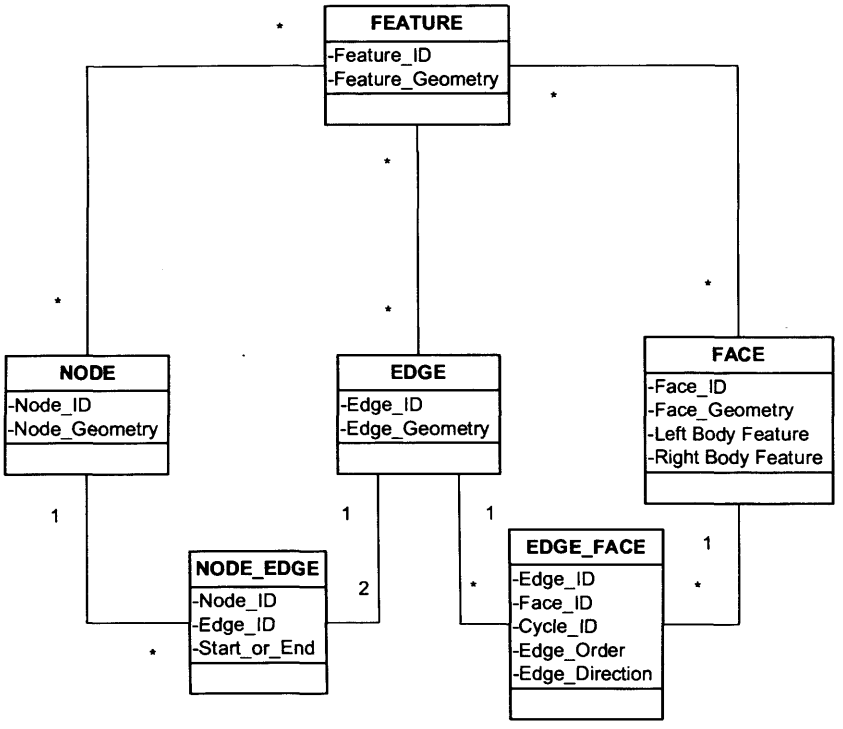


Figure 47 – Basic B-Rep Structure

In contrast, the approach shown in Figure 47 above simplifies the process of identifying the Edges associated with a particular Face and the Nodes associated with a particular Edge, using the EDGE_FACE and NODE_EDGE join tables. This in turn facilitates the identification of topological relationships between objects, a process which is based on the identification of shared primitives. It is also possible to use this information to reconstruct the Face - querying the EDGE_FACE table and sorting by EDGE_ORDER rapidly retrieves the required list. This approach starts from the Face, unlike the wing-edge approach which starts the search from the Node.

As with STS (Table 39), the reconstruction of an ordered list of the coordinates making up a Face from the Node primitives may not be required for 3DFDS. In an Object-Relational

environment, this information can also be stored directly as a spatial object associated with the Face itself, although this increases the maintenance overhead as coordinate information must be maintained in multiple locations.

In 3DFDS any primitives linked to an object are automatically part of that object, with exception tables handling containment exceptions. For example, when rebuilding a Body from the constituent VOLUMEs, any FACEs appearing twice can be ignored. This is similar to the approach taken when rebuilding the geometry of a Surface object from multiple constituent FACEs.

5.10.2 Extracting Coordinate Information in STS

Taking advantage of the Object-Relational environment, STS can store coordinate information in multiple ways, as shown in Table 39.

Option	*Advantages*	*Disadvantages*
Pure B-Rep - coordinate information only stored on Node primitive.	Coordinate information only stored once. Forces all objects to be manifold.	Difficult to reconstruct the object for metric and visualisation purposes. Cannot represent Edges made from multiple segments. Need to follow multiple joins to retrieve coordinate information.
As-Required – coordinate information only stored on the FEATURE	Coordinate information only stored once. Easy to maintain. Supports rapid metric queries and visualisation.	No access to primitive geometries to facilitate error correction.
Object-Relational 1 – coordinate information stored on Node, Edge and Face primitives.	Can represent multi-segment Edges. Do not need to follow multiple joins to find Face coordinates – information directly available. Takes advantage of Object-Relational database.	Reconstruction of the object for visualisation or metric query slow. Coordinate information occurs in multiple locations, complicating maintenance and requiring additional storage.
Object-Relational 2 – coordinate information stored on Node, Edge and Face primitives and on the FEATURE.	Supports topological queries and also rapid visualisation and metric query performance. Do not need to follow multiple joins to find Face coordinates – can determine information directly.	Coordinate information occurs in multiple locations, complicating maintenance and requiring additional storage.

Table 39 - Options for Extracting Ordered List of Face Coordinates

For STS, the second Object-Relational approach described is suggested, as having direct access to coordinate information for each primitive has advantages – as the primitives in question are simpler than the whole objects. Although having disadvantages in terms of additional storage requirements and data maintenance (due to redundancy as coordinate information is stored both against the FEATURE and against the primitives), it is possible to only store primitives of the dimension of the interior and the highest boundary dimension (see Table 26). This reduces the

volume of data in the TOPO_ tables and primitive tables, and improves query performance, as well as reducing the level of duplication of coordinate information reduced.

One final consideration is required when identifying the primitives associated with a particular Part object in STS to reconstruct the object for visualisation. This is due to the method utilised to handle containment exception situations, which are modelled by linking primitives not directly part of an object with that object, as shown, for example, in Table 32 above where Node N2 is linked to Surface B. Therefore, rules are required to ensure that contained primitives not forming part of the original object are not selected during any reconstruction of the object from the spatial representation of its primitives. If a primitive has a lower dimension than the (simple) part object with which it is associated, and the IS_BOUNDARY flag is false, then this depicts a containment situation and the primitive should not be used to reconstruct the object.

5.11 Summary

This Chapter presented the Simplified Topological Structure, optimised to improve query performance for binary topological queries. STS overcomes two issues with Extended 3DFDS, namely the number of relational joins and the requirement for exception tables. The structure consists of four primitive types (Node, Edge, Face and Volume) which are joined to the FEATURE via a series of TOPO_ tables. Each primitive join table excluding TOPO_VOLUME also has an associated IS_BOUNDARY flag, which determines whether the primitive forms part of the interior of a part object or not. The structure was reviewed in terms of its ability to model various characteristics of 3D objects including curved Surfaces and non-manifold objects, and also in terms of the determination of selected topological relationships including containment.

STS was then compared to the Extended 3DFDS described in Chapter 4, and also to the topological standard proposed by ISO 19107. Two potential limitations of STS were identified – the lack of left and right Volume links for Face primitives, and the lack of directed Edges. It is suggested that this information could be derived as required by the topological engine, using the spatial objects associated with the primitives. The performance impact of this approach, however, requires assessment as part of the engine development process.

Having selected three structures for the implementation of binary relationships (As-Required, STS and Extended 3DFDS) the following Chapter of this thesis describes the development of a dataset and a series of algorithms to underpin comparative performance testing.

6 The Test Dataset

Figure 48 - Overview of Document Structure showing Context of this Chapter

6.1 Introduction

In preparation for the description of system architecture and test design presented in Chapter 8, and the algorithms for 9-Intersection relationship determination reviewed in Chapter 7, this Chapter describes the dataset underpinning the tests, giving an overview of the data capture and replication processes required to populate all three data structures (As-Required Proxy, Extended 3DFDS and STS).

A description of the objects forming the dataset is first presented. This is followed by an overview of the tasks carried out to create and validate the dataset and within the context of the STS. Replication of this dataset to 1.08 million objects is described, and the processes to migrate this data from STS to 3DFDS and the As-Required structure are then reviewed, along with additional replication of the data within these structures. The Chapter concludes with a discussion reviewing the results of the dataset creation process, giving storage and index size requirements for each structure.

6.2 Dataset Description

The objects included in the test data are created from a comprehensive set of diagrams detailing possible 9-Intersection relationships in 3D between simple objects, documented by Zlatanova (2000, Chapter 6). Basing her work on that carried out by Egenhofer and Herring (1990), Zlatanova identifies a series of 25 conditions to eliminate topological relationships that are invalid between simple objects in a 3D context. Using these conditions, she has identified a series of relationships - summarised in Table 40 below - that form the basis of the dataset described here.

Dimension of Embedding Space	*Dimension of Objects*	*Number of Relationships*
1	Line and Line	8
2	Line and Line	33
3	Surface and Line	31
2	Surface and Surface	8
3	Body and Body	8
3	Surface and Surface	38
3	Body and Line	19
3	Body and Surface	19

Table 40 - Summary of Relationships identified by Zlatanova (2000)

Note that Line/Line relationships in 3D embedding space are not included in the original Zlatanova (2000) dataset as these are the identical to those in 2D.Figure 49 shows a sample of the source diagrams, specifically four of the Surface/Surface relationships.

Figure 49 – Examples of Surface/Surface Relationships (taken from Zlatanova 2000)

The R-Code values shown adjacent to each object pair are calculated as described in Chapter 3.

6.2.1.1 The Replicated Dataset

Line/Line relationships in 1D embedding space and Surface/Surface relationships in 2D embedding space were not included in the replicated dataset, as these are included in the higher dimension datasets (Line/Line in 2D and Surface/Surface in 3D). Additionally, the following relationships (Table 41) were not implemented for replication, due to issues that arose during their creation.

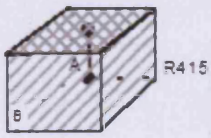
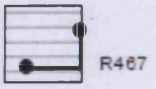
Relationship Group	*Relationship Diagram*	*Suggested Relationship*	*Issue Description*
Body/Line		R415	If A is entirely within B with only one end touching the surface (as the diagram would indicate), then the relationship diagrammed is not R415 but R476 – Int(A)∩ Ext(B) will return FALSE but Bnd(A)∩Int(B) will return TRUE. If the ends of line A touch the top and bottom of box B, the resulting relationship is R412, This relationship is already included in the dataset. If one end of A protrudes beyond the bottom of B the resulting relationship is R447 as INT A AND BND B INTERSECT AS TO INT A AND EXT B. This relationship is already diagrammed elsewhere.
Line/Surface		R467	This diagram shows relationship R499 as Boundary B does not intersect Interior A. R499 is already present in the dataset.
Body/Body		R511	This relationship was not created as part of the replicated dataset, as it could not be replicated automatically. The relationship was, however, captured manually as part of the data structure validation process.

Table 41 - 9-Intersection Relationships not Implemented for Replication

Thus the replicated dataset utilised for this project implements a total of 145 relationships out of the 164 identified in Table 40 (164 – (8 line/line +8 surface/surface + 3 problem diagrams)).

Additionally, thirteen of the Surface/Surface geometry pairings identified were not symmetric, generating differing results depending on the selection of A and B geometry (as shown in Figure 49). One pair of objects was created to represent both relationships and the relationship tested in both directions, giving a total of 132 object pairs.

6.3 *Creating the Datasets*

The creation of an automated topology engine for structure population is complex and beyond the scope of this thesis. Due to this complexity, initial attempts to populate STS involved writing SQL scripts to create the geometry and the populate required records in the TOPO_ tables, as well as those in the NODE, EDGE, FACE and VOLUME tables. Each pair of objects representing a relationship was created from scratch by manually encoding SQL INSERT statements, using coordinate values around the origin (0,0,0). As this totally manual approach

proved time-consuming and error prone a semi-automated method was developed to supplement this process. An overview of the structure population process for STS is given in Figure 50 below. A full description of the structure population processes for all three structures is given in Appendix 4.

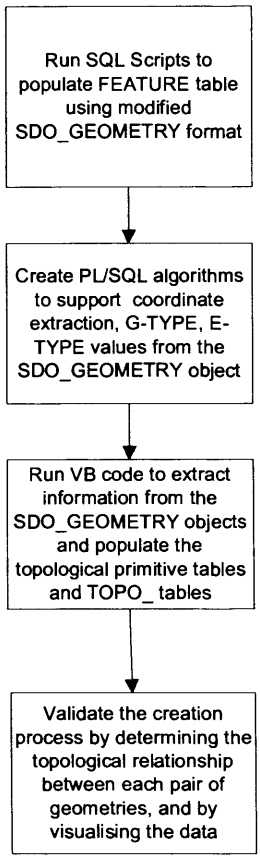


Figure 50 - Populating the STS Data Structure

In the absence of a topology engine, a modified version of the Oracle SDO_GEOMETRY data type was used to manually encode the topological relationships, using a variety of non-standard codes and repetition of primitives to represent containment relationships. An additional field called COLLECTION_TYPE was also added to the FEATURE table to distinguish between closed surfaces and true 3D solids. SQL scripts were written and run to insert the modified SDO_GEOMETRY objects, and an algorithm written to decode the SDO_GEOMETRY and create the primitives by extracting coordinate information from the modified SDO_GEOMETRY. To take into account the fact that the dataset was required for both the 3DFDS and STS structures, an EDGE primitive was defined as a single line segment.

Once the dataset was created and validated (though the use of SQL queries and custom-built visualisation tools), it was replicated 8, 64, 512 and 4096 times to provide datasets of varying size to underpin the process of determining scalability, storage and performance testing. This

was done by calculating the three extents of the dataset (in the X, Y and Z directions), and then duplicating and shifting the dataset using the extents as guidance for the shift distance. The dataset was replicated in the X, Y and Z directions individually, then in XY, YZ and XY, and finally in XYZ. Each process therefore resulted in a dataset eight times the previous size.

Given that the primitives for the STS and 3DFDS structures are identical, the latter were created by use of a simple copy process. Hierarchical relationships between primitives (Node/Edge/Face/Volume), which are encoded directly in the 3DFDS structure, were then calculated through a process of coordinate comparison and the resulting data replicated as described for STS.

Finally, the use of R-Tree indexes for the tests for the As-Required structure requires that the SDO_GEOMETRY objects for this structure are created correctly and that no non-standard codes are included in the objects. Thus the custom SDO_GEOMETRY objects created for STS population could not be re-used for this purpose, and corrected objects were created by querying the primitives of the STS structure and rebuilding the SDO_GEOMETRY. Six copies of the resulting datasets were made to allow for multiple R-Tree index tolerances values to be tested (as per Preliminary Test 3 described in Chapter 8). R-Tree indexes were also created for each of the primitive tables in STS and 3DFDS, although they could not be created for the original FEATURE tables for these datasets due to the modified SDO_GEOMETRY.

6.4 The Resulting Datasets

Images of the complete set of data created by the above process can be found on the attached CD, with sample images in Appendix 4. There are two main elements to storage requirements within a database – the data itself, and any associated indexes. Processes and SQL queries to determine the storage requirements for each structure are described in Appendix 4. The results are presented here.

6.4.1 The Datasets

The largest dataset contains a total of 1,081,344 FEATURE records. Additionally, interim datasets have also been maintained to support algorithm complexity testing. The extents of each dataset are given in Table 42 below. As can be seen, the values for maximum Y and Z values are much lower than those for X. This is due to the initial translation of FEATURE records in the X direction as part of the data structure population process - each object pair was created around the origin (0,0,0) and then trans-located along the X axis to avoid overlaps between object pairs.

	Min X	*Min Y*	*Min Z*	*Max X*	*Max Y*	*Max Z*
* 1	6	0.5	0	1318.0	5.5	6.0
* 8	6	0.5	0	2630.0	11.0	12.0
* 64	6	0.5	0	5254.0	22.0	24.0
* 512	6	0.5	0	10502.0	44.0	48.0
* 4096	6	0.5	0	20998.0	88.5	96.0

Table 42 - Extents of Replicated Datasets

6.4.1.1 *As-Required Data*

Table 43 gives values for index tolerances for each of the As-Required tables. The following tables give the resulting storage values (in MB) for each index tolerance at each level of replication.

Table Name	*Index Tolerance (m)*
GEOMETRY_SDO_RELATE_005	0.05
GEOMETRY SDO RELATE	0.5
GEOMETRY_SDO_RELATE_1	1
GEOMETRY_SDO_RELATE_5	5
GEOMETRY_SDO_RELATE_100	100
GEOMETRY_SDO_RELATE_500	500

Table 43 - Index Tolerances for As-Required Data

	Total Number of Records	*Table Storage (MB)*	*Non-Spatial Index Storage (MB)*	*Spatial Index Storage (MB)*	*Total Storage (MB)*
* 1	264	0.0730	0.0034	0.0238	0.1002
* 8	2,112	0.6000	0.0280	0.2361	0.8641
* 64	16,896	4.8100	0.2320	2.1441	7.1861
* 512	135,168	38.6700	1.9200	18.1537	58.7437
* 4096	1,081,344	316.5900	15.5400	164.5657	496.6957

Table 44 - Storage for As-Required Data – Index Tolerance 0.05m

	Total Number of Records	*Table Storage (MB)*	*Non-Spatial Index Storage (MB)*	*Spatial Index Storage (MB)*	*Total Storage (MB)*
* 1	264	0.0730	0.0034	0.0238	0.1002
* 8	2,112	0.6000	0.0280	0.2361	0.8641
* 64	16,896	4.8100	0.2320	1.9427	6.9847
* 512	135,168	38.6700	1.9200	25.1622	65.7522
* 4096	1,081,344	316.5900	15.5400	164.5657	496.6957

Table 45 - Storage for As-Required Data – Index Tolerance 0.5m

	Total Number of Records	*Table Storage (MB)*	*Non-Spatial Index Storage (MB)*	*Spatial Index Storage (MB)*	*Total Storage (MB)*
* 1	264	0.0730	0.0034	0.0303	0.1067
* 8	2,112	0.6000	0.0280	0.2275	0.8555
* 64	16,896	4.8100	0.2320	1.9774	7.0194
* 512	135,168	38.6700	1.9200	25.1622	65.7522
* 4096	1,081,344	316.5900	15.5400	164.5657	496.6957

Table 46 - Storage for As-Required Data – Index Tolerance 1 m

	Total Number of Records	*Table Storage (MB)*	*Non-Spatial Index Storage (MB)*	*Spatial Index Storage (MB)*	*Total Storage (MB)*
* 1	264	0.0730	0.0034	0.0238	0.1002
* 8	2,112	0.6000	0.0280	0.2361	0.8641
* 64	16,896	4.8100	0.2320	2.1441	7.1861
* 512	135,168	38.6700	1.9200	18.1537	58.7437
* 4096	1,081,344	316.5900	15.5400	164.5657	496.6957

Table 47 - Storage for As-Required Data – Index Tolerance 5 m

	Total Number of Records	*Table Storage (MB)*	*Non-Spatial Index Storage (MB)*	*Spatial Index Storage (MB)*	*Total Storage (MB)*
* 1	264	0.0730	0.0034	0.0303	0.1067
* 8	2112	0.6000	0.0280	0.2318	0.8598
* 64	16896	4.8100	0.2320	1.8686	6.9106
* 512	135168	38.6700	1.9200	15.1327	55.7227
* 4096	1081344	316.5900	15.5400	120.9134	453.0434

Table 48 - Storage for As-Required Data – Index Tolerance 100 m

	Total Number of Records	*Table Storage (MB)*	*Non-Spatial Index Storage (MB)*	*Spatial Index Storage (MB)*	*Total Storage (MB)*
* 1	264	0.0730	0.0034	0.0303	0.1067
* 8	2,112	0.6000	0.0280	0.2318	0.8598
* 64	16,896	4.8100	0.2320	1.8686	6.9106
* 512	135,168	38.6700	1.9200	15.1327	55.7227
* 4096	1,081,344	316.5900	15.5400	120.7551	452.8851

Table 49 - Storage for As-Required Data – Index Tolerance 500 m

6.4.1.2 STS and 3DFDS

Table 50 and Table 51 give the total number of records for each dataset, including all the entries in the TOPO_ join tables as well as those in the topological primitive tables, FEATURE, and

TOPO_PART_TABLE (as the FEATURE tables in STS and 3DFDS could not be indexed, values for R-Tree index size have been taken from the As-Required dataset, which represents a corrected set of FEATURE geometry). Additionally, the total storage required for data and for indexes (both spatial and non-spatial) is listed. Storage statistics (all given in MB) were obtained following Oracle's (Oracle, 2006g) recommended procedures described in Appendix 4.

	Total Number of Objects	*Total Number of Records including all Primitives and Join Tables*	*Table Storage (MB)*	*Non-Spatial Index Storage (MB)*	*Spatial Index Storage (does not include index on the main FEATURE Table) (MB)*	*Total Storage (MB)*
* 1	264	8,427	0.5694	0.2840	0.4288	1.2823
* 8	2,112	70,388	4.8449	2.3728	6.1574	13.3750
* 64	16,896	540,224	39.1742	20.0617	29.0043	88.2403
* 512	135,168	4,321,792	316.8457	161.6447	299.0551	777.5455
* 4096	1,081,344	34,574,336	2576.4336	1059.1069	2571.5170	6207.0574

Table 50 - Record Count and Storage for STS

	Total Number of Objects	*Total Number of Records including all Primitives and Join Tables*	*Table Storage (MB)*	*Non-Spatial Index Storage (MB)*	*Spatial Index Storage (does not include index on the main FEATURE Table) (MB)*	*Total Storage (MB)*
* 1	264	10,967	0.6056	0.3730	0.4288	1.4074
* 8	2,112	87,722	5.0691	3.0010	6.1574	14.2275
* 64	16,896	700,164	41.7492	26.1927	29.0043	96.9462
* 512	135,168	5,613,914	336.1537	212.3276	299.0551	847.5364
* 4096	1,081,344	44,899,024	2751.9538	1759.5503	2571.5170	7083.0210

Table 51 - Record Count and Storage for 3DFDS

6.4.2 Comparing the Resulting Datasets

As can be seen from the dataset sizing results, 3DFDS storage requirements are greater than those for STS for all replicated datasets, with a difference of 876 MB for the 1.08 million record dataset. As the primitives in both cases are identical, this is due to the additional records required to maintain the NODE_EDGE, EDGE_FACE and FACE_VOLUME relationships, and is split between the Table Storage difference (175.52 MB) and the Non-Spatial Index Storage difference (700 MB) . The presence of these join tables does, however, remove the requirement for an SDO_GEOMETRY representation of the Edge or Face primitives (if single segment Edges are assumed) as these can be reconstructed from the Node coordinates. If 3DFDS is implemented with all coordinates being derived from those stored against the Node primitive

(i.e. the geometry is reconstructed rather than stored with the object record) then the total storage required for the Node table is 742 MB. This still exceeds the total storage requirement for the 'As-Required' structure (due to the higher number of Node primitives – 5.2 million as opposed to 1.08 million FEATURE records).

Due to the pair-wise dataset, the total storage for both 3DFDS and STS is likely to be higher than that for the same number of real-world features, as in the latter case, many more primitives would be shared. Further tests using real-world data are required to determine if this will reduce storage more significantly 3DFDS. For example, in 3DFDS if a Node is a primitive of 3 objects it is only created once (although it might be referenced by three records in the NODE_EDGE table). For STS, three records will be created in the TOPO_NODE table.

For the As-Required structure, storage requirements drop as index tolerance increases, due to the fact that more objects are associated with the same node on the R-Tree index – more objects on a single the R-Tree index implies that fewer index nodes are created. The index tolerance distance determines the number of objects associated with the same leaf of an R-Tree index. The smaller the tolerance, the fewer objects on the leaf. This has advantages as fewer candidate objects are passed to the topological engine following filtering, but also increases index size. Storage requirements for this structure are significantly lower than those for either 3DFDS or STS, as the latter two include both the original object SDO_GEOMETRY objects and those associated with the primitives forming part of the topological structure.

The population algorithm was simplified by splitting all multi-segment lines into many Edges, and all multi-part surfaces into many Faces. This has, however, resulted in the creation of additional records in the TOPO_ join tables and in the primitive tables. For example, in Figure 51(a) five Edge primitives are used to represent the boundary of Face F1 instead of the two in Figure 51(b). Similarly, three Edge primitives represent Face F2 instead of two. This results in an artificially inflated number of records, increased storage requirements and reduced query performance, although it does perhaps better emulate the expected number of primitives to be associated with real-world features.

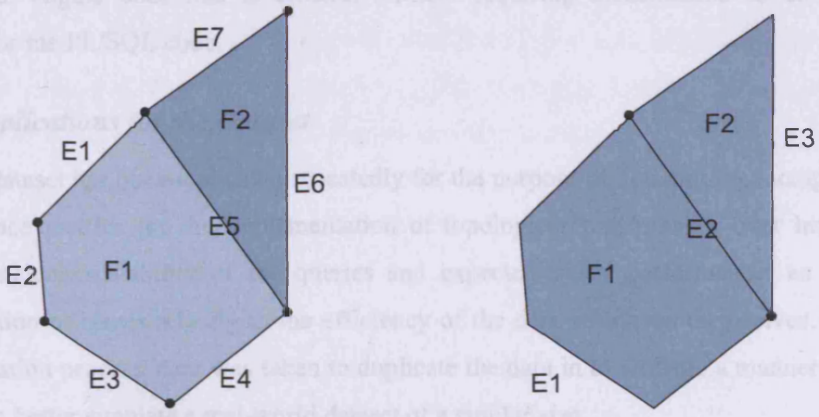


Figure 51 - Additional Edge Primitives created due to Single-Segment Edge Requirement

In general, a trade off is required when choosing between simple (single-segment Edges, triangular Faces, tetrahedral Volumes) and complex (multi-segment Edges, polygon Faces, polyhedral Volumes) primitive types. In the former case, primitives are easier to identify and maintain, there is no requirement to validate Face planarity and the primitives can also be utilised to visualise the 3D objects using existing 3D visualisation algorithms. It is also possible to store coordinate information only once, on the Node primitive. However, a higher number of primitives are required for object representation, reducing 9-Intersection query performance due to increased table and index sizes. In the latter case (non-simplex primitives), data structure maintenance time may increase, multiple copies of the coordinate information are required, but relationship query times are reduced. STS can support both approaches. However, implementing multi-segment Edges in the Extended 3DFDS structure is only possible if the requirement to build the Edge geometry from Nodes is removed (i.e. Edge geometry is stored alongside the Edge primitive).

The structure population approach also assumes that all levels of primitives are required for relationship determination. However, as described for STS (Section 5.2.3), it may be possible to utilise primitives of the dimension of the interior and the highest boundary dimension only. Thus in Figure 51, the surface could be represented by the Faces and Edges, without the Nodes. This would yield an identical 9-Intersection result and would reduce the volume of data in the primitive tables and the TOPO_ tables, and thus improve query performance. The data maintenance task would also be simplified. The only exception to this case would be when a lower level primitive is the only primitive shared with another object. As the 9-Intersection relationships focus solely on shared primitives, this more topologically correct approach reducing the number of primitives and limiting the level of primitives stored can be taken by the

topological engine once this is created, without requiring modification to either the data structure or the PL/SQL code.

6.5 *Applications for the Dataset*

The test dataset has been replicated repeatedly for the purpose of determining storage and query performance metrics for the implementation of topological functionality over large datasets. Metrics include scalability of the queries and expected query performance, as well as the identification of issues relating to the efficiency of the data structures themselves. Throughout the replication process, care was taken to duplicate the data in as realistic a manner as possible, in order to better simulate a real-world dataset of a similar size.

A number of additional applications can be identified for this dataset. It provides a basis to systematically validate the correct implementation topological query functionality in relation to 9-Intersection relationships. As the dataset is comprehensive, containing the relationships identified by the framework for simple objects in 3D, it is far more useful for this purpose than a real-world 3D urban dataset, where many relationships may not be represented (although this would provide a wider variety of examples of the same relationship, see Section 11.4.6). The dataset has been purpose-built for relationship verification, and each query will therefore give predictable, consistent results which can be verified against the original data. This predictability is difficult to achieve using a real world dataset where the researcher may not be familiar with each and every object and hence with the results expected from the topological queries.

The dataset can provide a basis for the testing of the implementation of other topological frameworks. Relationships such as the Clementini *et al.* (1993) and the Set Theoretic or Boolean (OGC 2006) have both been suggested as topological standards by the OGC. These require knowledge of interior, boundary and exterior of objects, given by both STS and 3DFDS. Applications to other frameworks such as the Dimensional Model can also be considered.

6.6 *Summary*

This Chapter provided a brief overview of the dataset creation process for the to be used for comparative performance testing in this research, describing the creation and replication processes used to populate each structure with between 264 and 1.08 million objects. The outcome in terms of storage requirements for both table and index data was then reviewed and compared. It was concluded that the As-Required structures require significantly less storage than the topological data structures. Comparing STS and 3DFDS reveals that STS requires less storage if it is assumed that both structures will include SDO_GEOMETRY objects in association with Node, Edge and Face primitives.

Potential applications, including validating other topological data structures and algorithms, were also identified for the test dataset. The following Chapter describes algorithms implemented in PL/SQL to query the 9-Intersection relationships modelled by this dataset.

7 Algorithm Design

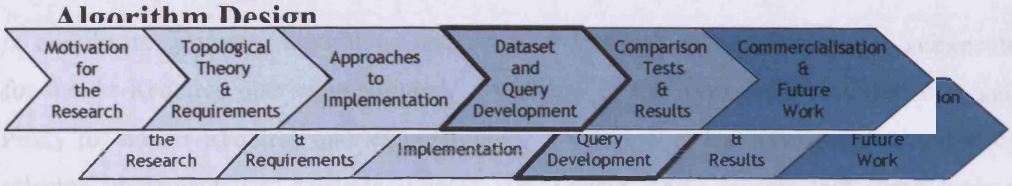


Figure 52 - Overview of Document Structure showing Context of this Chapter

7.1 Introduction

As described in Chapter 3, three types of queries can be derived from the review of requirements for topology in 3D, namely:

- Determine 9-Intersection relationship between two objects (abbreviated to *9-Intersection Pairs*).
- Find any objects intersecting with a given object (abbreviated to *Find Intersecting Objects*).
- Find any objects having a specific 9-Intersection relationship with a given object (abbreviated to *Find Objects with Relationship*).

The identification of three query types for implementation greatly simplifies the interface presented to the developer of a 3D GIS system or to the specialist end-user with knowledge of SQL.

Three approaches to relationship determination have been selected for comparison – an As-Required approach and two structures – STS and Extended 3DFDS. A description of the approach taken for the As-Required queries and their implementation is given first. Existing approaches to the identification of 9-Intersection relationships using topological data structures are then reviewed, and an alternative set-based approach is proposed to overcome some of the limitations of these. Design and implementation of the query algorithms is then described in the context of the structures. To ensure a fair basis for comparison, optimal query performance is required for both the STS and the 3DFDS algorithms. A description of the review process followed to ensure this concludes the Chapter.

The implementation process described here has been presented in terms of the selected environment, namely an Oracle 10g database (Oracle 2006). However, it should be emphasised that it is possible to implement both the data structures described in Chapters 4 and 5 and these algorithms in any Object-Relational database supporting spatial data types, spatial indices and procedural extensions to SQL.

7.2 As-Required Queries

In the absence of a topological engine, which would provide functionality to execute the complex coordinate geometry algorithms and return 9-Intersection relationships, a suitable Proxy for the As-Required queries is required. A review of the functionality offered by the selected implementation database (Oracle 10g, Oracle 2006) reveals that some limited 3D functionality is available.

7.2.1 3D Filtering and Proxy Selection

Spatial data in Oracle is stored in an Object-Relational format (known as an SDO_GEOMETRY object), with no intrinsic topology. This storage mechanism supports simple and complex Point, Line and Polygon data, and also allows data to be stored and indexed using 3D coordinates. The format allows the creation of an R-Tree index to improve query performance. Metadata is associated with each SDO_GEOMETRY field to allow the user to define the extents and projection of the dataset. This is then used for R-Tree index creation.

At the time of writing, Oracle Spatial supports 3D indexing and primary filter querying (using a 3D R-Tree). Filtering provides a first-pass answer to a query, reducing the list of objects against which advanced computational algorithms must be applied. The SDO_FILTER (Oracle 2006d) operator (described by Oracle as a *primary* filter, but also known as broad-phase filtering) makes use of the R-Tree index on the data to identify objects that potentially intersect or have some other topological relationship with a given object, or to determine if two objects are likely to intersect. The query returns a list of potentially intersecting objects. These are then be passed through to a secondary filtering query (also known as narrow phase filtering, in this case undertaken by the topological engine) to determine the exact nature of the relationship.

Figure 53 - Oracle Filtering Process (Oracle 2006d)

Although the SDO_FILTER option is available for 3D data in the current implementation of Oracle (2006d), the secondary filter (known as SDO_RELATE) has not yet been implemented in 3D. However, given the two-staged approach shown in Figure 53 the SDO_FILTER provides the best available Proxy for As-Required relationship calculation. The structure of the SDO_FILTER function call is shown in Table 52 below.

Function Name	*Description*	*Parameters*	*Returns*
SDO_FILTER	Runs a first pass query to identify candidate intersecting objects using an R-Tree Index	One or two FEATURE_IDs Additional parameters include the option to specify the minimum and maximum resolution (i.e. dimension) of any returned objects. This option is not available in the 3D context.	If one FEATURE_ID provided, returns a list of candidate intersecting FEATURE_IDs. If two objects listed, returns 'TRUE' if an intersection occurs (i.e. if the objects are within the given index tolerance of each-other).

Table 52 - Oracle's SDO_FILTER Operator

Two queries were implemented using this function. Firstly, a query to find any objects intersecting with a given object was written. This acts as a Proxy for the *Find Intersecting Objects* and *Find Objects with Relationship* query described in Chapter 3. The second query takes two given FEATURE_IDs and validates their potential for intersection, giving a Proxy for the *9-Intersection Pairs* query.

Proxy for Find Intersecting Objects and Find Objects with Relationship

```
SELECT * FROM FEATURE C
WHERE SDO_FILTER(C.FEATURE,
SELECT B.FEATURE FROM FEATURE_SDO_RELATE B
WHERE B.FEATURE_ID = << THE FEATURE_ID>>)) = 'TRUE'
```

Proxy for 9-Intersection Pairs

```
SELECT B.FEATURE_ID
FROM FEATURE A, FEATURE B
WHERE SDO_FILTER(A.FEATURE, B.FEATURE) ='TRUE'
AND A.FEATURE_ID = << FEATURE_ID 1>> AND B.FEATURE_ID = << FEATURE_ID 2>>
```

Proxy results are further improved by running both primary and secondary filter queries against a 2D dataset and recording differences in execution time. As the secondary queries in 2D do return full relationship details, these differences provide an indication of the execution time required for the coordinate geometry algorithms in 3D. For example, the following calculations can be applied to improve the Proxy results obtained for the *9-Intersection Pairs* queries.

$$3D\_9I\_Pairs = (3D\_Filter + (2D\_9I\_Pairs - 2D\_Filter))$$

Equation 1 - Improving the Proxy

The Proxy can be further improved by considering the expected difference between the number of 2D and 3D point-to-polygon distance (see Section 8.4.4).

7.3 *Determining 9-Intersection Components*

A number of approaches to support the determination of 9-Intersection relationships against a topological data structure can be identified.

7.3.1 Determining Find Objects with Relationship using minimal subsets

Clementini *et al.* (1994) present an algorithm that uses the minimal subset of information required to identify/distinguish the specific relationship from all the other possible relationships, and describe this for the eight 9-Intersection relationships between two simple objects. For example, for the DISJOINT relationship, knowledge that the interior/interior and the boundary/boundary relationships are both empty is enough to identify the relationship. A series of conditions can be developed to rapidly identify object pairs having a specific relationship.

7.3.2 Determining *9-Intersection Pairs* Using Decision Trees

Clementini *et al.* (1994) present two decision trees for the rapid evaluation of the 9-Intersection relationships between simple region objects. These trees can be developed based on a series of conditions – for example, for two simple region objects, if the intersection of the boundary components is null, then there is no need to evaluate the intersection of the interior of one with the interior of the other, as this information can be deduced from the interior/boundary relationships. If the boundary of A does not intersect the interior of B AND the boundary of B intersects the interior of A then this is a contains relationship. If this is the case, the interior of A will always intersect the interior of B.

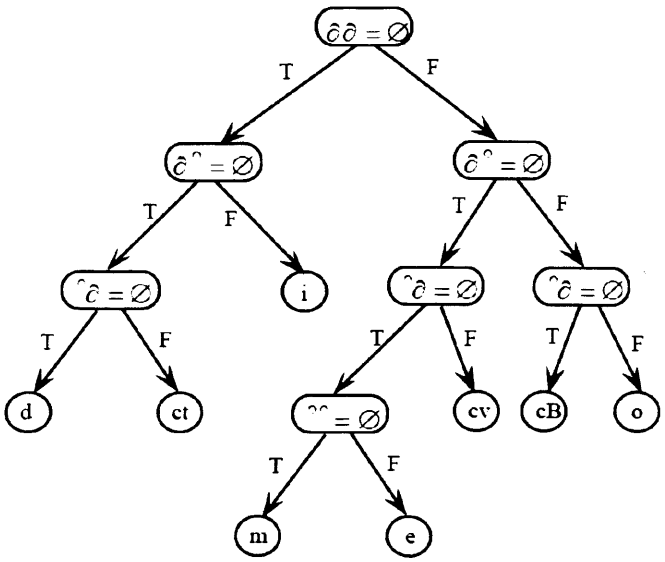


Figure 54 - Decision Tree for Region/Region Relations based on Native Cost Model

The authors use a native cost model (assuming that all relations occur with equal possibility, Figure 54) and a refined cost model (which takes into account possible distributions of the

relationships) to develop these trees. The work has been carried out in a 2D context and for simple, single-part objects.

7.3.3 *9-Intersection Pairs* in 3D using Conditions

As is described in Chapter 6, Zlatanova (2000, Chapter 6) has generated a comprehensive set of diagrams to represent the possible 9-Intersection relationships in a 3D context. In order to validate the completeness of the dataset, she provides a series of conditions which can be used for the implementation of 9-Intersection relationship determination. These bridge the gap between theoretical relationships and those that can occur in practice, and use of a series of conditions built on the approach described for 2D data by Egenhofer and Herring (1990) and Egenhofer and Franzosa (1991).

The conditions (25 in total) can be utilised to eliminate impossible intersections when deriving 9-Intersection relationships. 9-Intersection relationships can be determined provided that the following factors are also known:

- The dimension of the object.
- The connectivity of their boundaries.
- The dimension of the embedding space.

These conditions limit the number of intersection operations that are required between the primitives representing the interior and boundary of each object. For example, if the objects have equal dimension, non-zero co-dimension and disconnected boundaries, then only 6 conditions out of the 25 need to be tested. A hierarchy of conditions can thus be created, using a similar approach to that described by Clementini *et al.* (1994). Using conditions can greatly reduce the amount of query processing time required to identify the relationship between two objects.

7.3.4 Reviewing the Approaches

The approaches described above present a minimal set of calculations required to determine the *9-Intersection Pairs* relationship between two objects, and in a general 2D setting have been proved to be efficient. Schneider and Behr (2006) also present a list of conditions for relationships between complex and compound objects in 2D. Further work is, however, required to develop the tree and constraint-based approaches described above for 3D, complex, multi-part objects and to determine if in fact the use of decision trees will aid relationship determination performance.

Additionally, as Zlatanova (2000, pg. 126) states, using conditions may result in a strict enforcement of the rules where this is not necessary, resulting in the omission of a number of

possible relationships. The approach described also results in differing numbers of condition tests for the various relationships between objects – i.e. an unbalanced decision tree. This in turn may result in inconsistent query performance across the whole spectrum of relationships.

Wei *et al.* (1998) give an example of where a strict enforcement of conditions may fail, noting that a closed line object does not have a boundary in topological terms. This implies that intersections between such objects may return anomalous results according to the above conditions. A object may intersect with the interior and the exterior of the closed line without intersecting with its boundary, violating Condition 9 in Zlatanova (2000) which states that "If A's interior intersects with B's interior and exterior then it must intersect with B's boundary and vice versa".

7.4 9-Intersection Relationships using Set Operators

An alternative approach utilises the set operators INTERSECT and MINUS on the primitives representing the interior and boundary of each object, directly testing each component of the 9-Intersection relationship rather than deriving the value through the application of conditions.

An overview of the algorithms used to determine the 9-Intersection relationships between Part Objects and the corresponding relationships between whole objects is presented here. Note that in the following discussion, the two Part Objects form part of separate parent objects.

7.4.1 Part Objects

Interior of Part object A intersecting with Boundary of Part object B (and vice-versa)

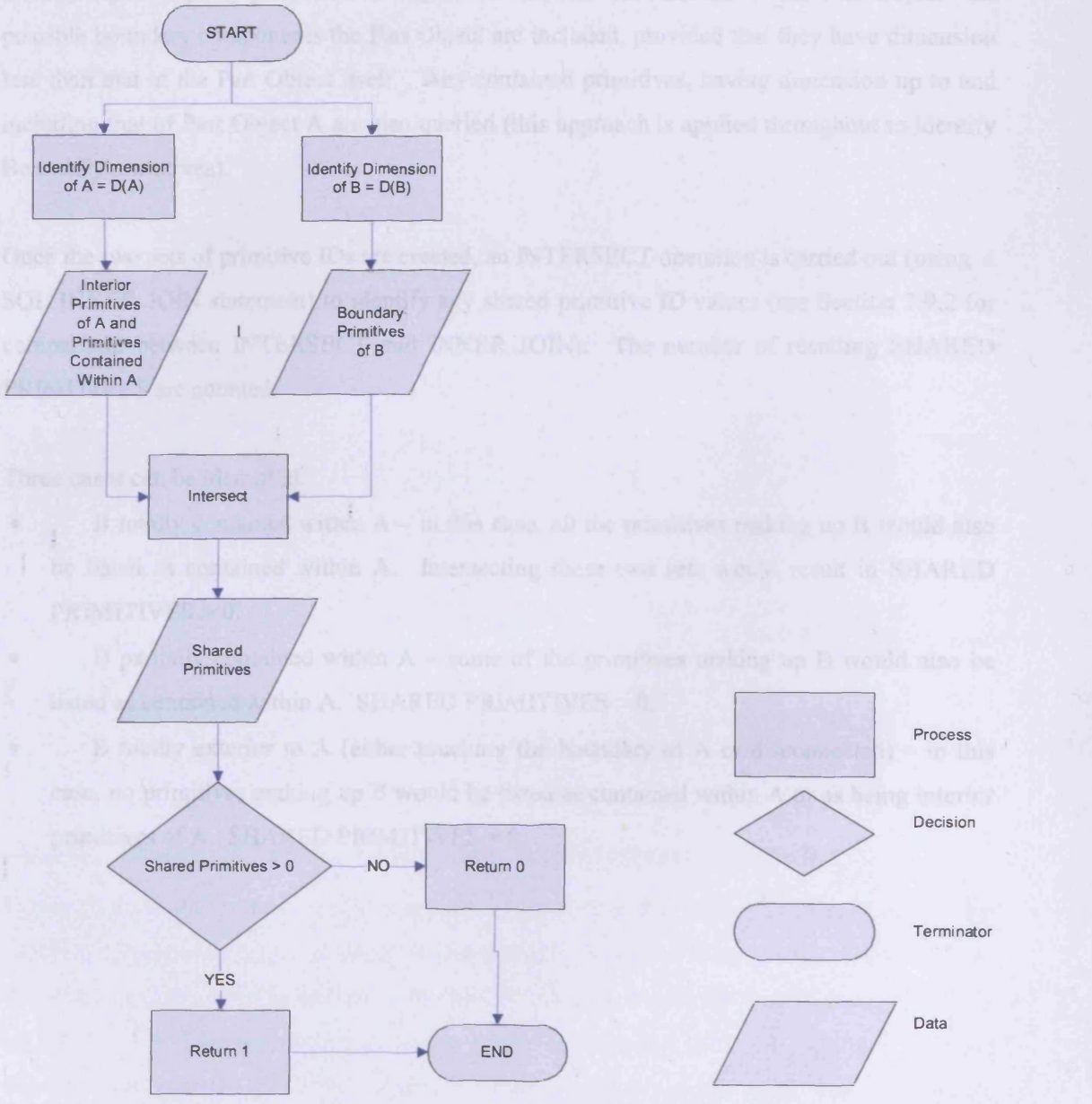


Figure 55 - Interior of Part object A Intersecting with Boundary of Part object B

Figure 55 shows a flow diagram for the algorithm to determine the intersection of the interior of one Part Object with the boundary of another. This function returns TRUE if an interior primitive of A is also an interior primitive of B. The algorithm does not assume that all objects will be associated with Nodes. Nodes, Edges, Faces and Volumes are all tested as these can

represent interior primitives of the Part Objects, depending on the dimension of the Part Object (identified as Dim(A) and Dim(B) in the first step of the flow diagram).

Similarly, the algorithm does not assume that the dimension of the boundary of a Part Object is directly represented by primitives having dimension one less than that of the Part Object - all possible boundary components the Part Object are included, provided that they have dimension less than that of the Part Object itself. Any contained primitives, having dimension up to and including that of Part Object A are also queried (this approach is applied throughout to identify Boundary primitives).

Once the two sets of primitive IDs are created, an INTERSECT operation is carried out (using a SQL INNER JOIN statement) to identify any shared primitive ID values (see Section 7.9.2 for comparison between INTERSECT and INNER JOIN). The number of resulting SHARED PRIMITIVES are counted.

Three cases can be identified:

- B totally contained within A – in this case, all the primitives making up B would also be listed as contained within A. Intersecting these two sets would result in SHARED PRIMITIVES > 0.
- B partially contained within A – some of the primitives making up B would also be listed as contained within A. SHARED PRIMITIVES > 0.
- B totally exterior to A (either touching the boundary of A or disconnected) – in this case, no primitives making up B would be listed as contained within A or as being interior primitives of A. SHARED PRIMITIVES = 0.

Boundary of Part object A interesting with Exterior of Part object B (and vice versa)

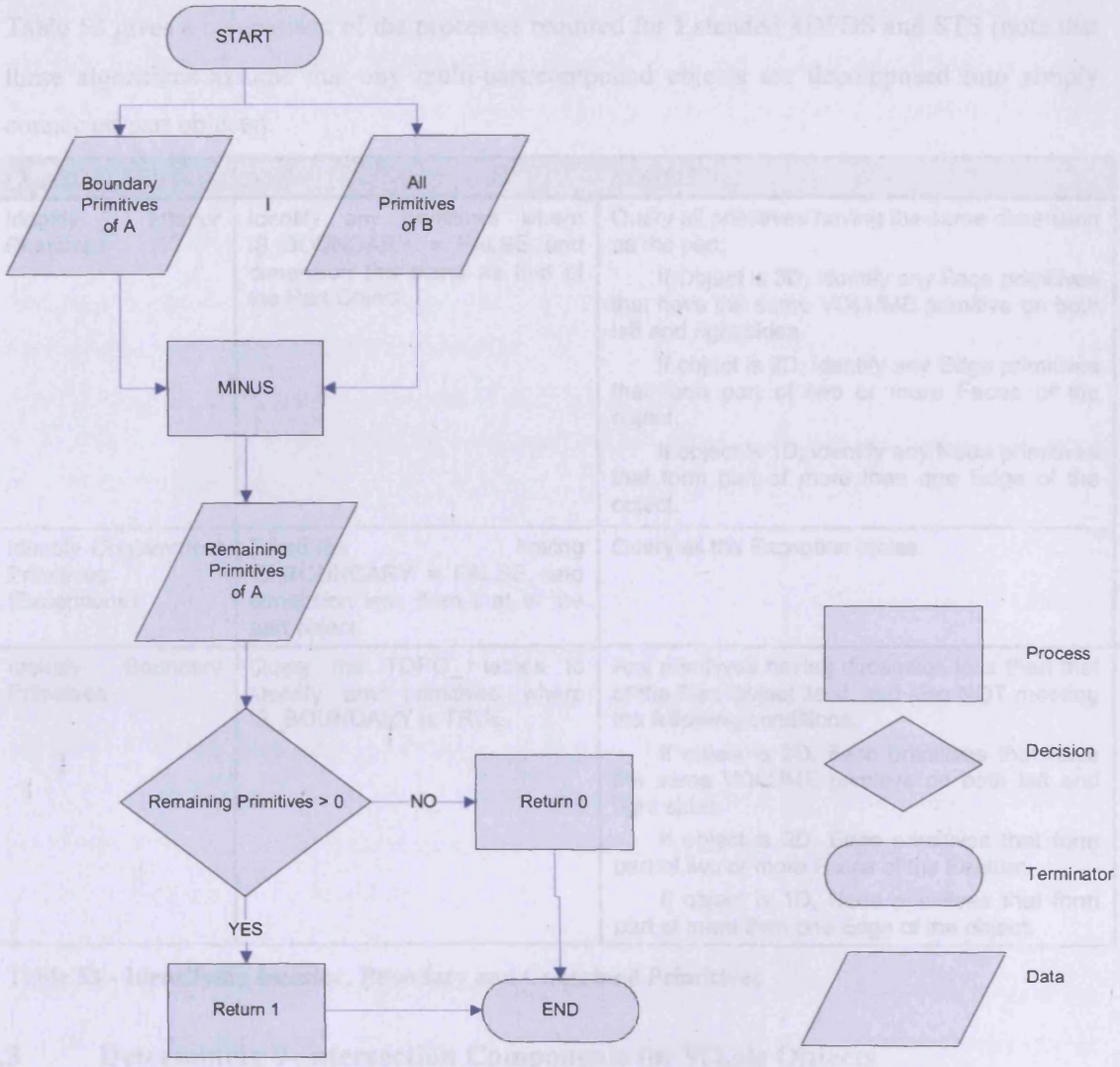


Figure 56 - Boundary of Part object A intersecting with Exterior of Part object B

Figure 56 diagrams the process to determine the boundary of Part object A intersecting with the exterior of Part object B or vice versa. In this case, the exterior of B is represented by taking all the primitives making up B and using the MINUS operator to subtract them from the boundary primitives of A. This algorithm will always return TRUE unless ALL the primitives of B are also boundary primitives of A. Again, no assumptions are made as to which primitives constitute the boundary of the object – this information is encoded in the data structure itself.

7.4.2 Identifying Boundary, Interior and Contained Primitives

Table 53 gives a comparison of the processes required for Extended 3DFDS and STS (note that these algorithms assume that any multi-part/compound objects are decomposed into simply connected part objects).

Query	*STS*	*3DFDS*
Identify Interior Primitives	Identify any primitives where IS_BOUNDARY = FALSE and dimension the same as that of the Part Object.	Query all primitives having the same dimension as the part. If Object is 3D, identify any Face primitives that have the same VOLUME primitive on both left and right sides. If object is 2D, identify any Edge primitives that form part of two or more Faces of the object. If object is 1D, identify any Node primitives that form part of more than one Edge of the object.
Identify Containment Primitives (Exceptions)	Primitives having IS_BOUNDARY = FALSE, and dimension less than that of the part object.	Query all the Exception tables.
Identify Boundary Primitives	Query the TOPO_ tables to identify any primitives where IS_BOUNDARY is TRUE.	Any primitives having dimension less than that of the Part Object itself, and also NOT meeting the following conditions: If object is 3D, Face primitives that have the same VOLUME primitive on both left and right sides. If object is 2D, Edge primitives that form part of two or more Faces of the Feature. If object is 1D, Node primitives that form part of more than one Edge of the object.

Table 53 - Identifying Interior, Boundary and Contained Primitives

7.4.3 Determining 9-Intersection Components for Whole Objects

In general, these algorithms build on those described for part-objects, iterating through each Part of object A and identifying its relationship with each Part of object B. In the case of Interior and Boundary intersections, the algorithm assumes that if the intersection is TRUE for one part A/ part B, then it is TRUE for the whole object. This builds on the approach described by Clementini *et al.* (1995).

For intersections involving the exterior of objects, an alternate approach is required. Iterating over the relationships between individual Parts of A and Parts of B may not yield a correct result, as it is possible for the boundary of one Part of object A not to intersect one Part of object B but to intersect another Part. Unlike interior and boundary primitives, the exterior of objects is not represented in the data structure. For the intersection of the boundary of A with the exterior of B to be TRUE for a multi-part object, at least one of the boundary components of object A should not form part of B or intersect with B. Therefore the algorithm first finds all the

boundary primitives related to object A (by joining the FEATURE table with the TOPO_PART_TABLE), and then subtracts all the primitives associated with object B (again, identifying these through the TOPO_PART_TABLE). If any primitives remain, then the intersection is TRUE. These 'whole object' algorithms return identical results to the 'part object' algorithms should an object only have one associated part.

7.4.4 Reviewing the Set-Based Approach

The algorithms described here utilise the following information to determine the required components of the 9-Intersection relationship:

- A list of the boundary primitives of each object (if any).
- A list of the interior primitives of each object Along with any primitives contained within the feature.
- A list of any primitives contained within each object.
- The dimension of each object.

Additionally, they make two assumptions about 9-Intersection matrix components – the exterior of A will always intersect that of B and all part objects are simply connected. This implies that any non-boundary primitive having dimension less that that of the part object is a containment exception. In STS, boundary information is encoded directly into the data structure using the IS_BOUNDARY flag. In 3DFDS, the information is derived by assuming that interior primitives have the same dimension as the object (or part object) itself, and that boundary primitives have dimension less than the object.

To overcome the issues with condition-based approaches, all nine intersection components are evaluated when using set operators. This provides greater flexibility when determining relationships, and does not enforce a particular series of conditions which in turn may constrain the object types whose 9-Intersection relationships can be determined. However the advantage of condition-based algorithms lies in the ability to rapidly determine relationships without evaluating each component. Full component evaluation may thus impact algorithm performance.

Implementing relationship identification between part objects allows users to determine relationships between individual components of objects as well as between the objects as a whole. Relationships between complex objects (having holes, cavities or tunnels) can also be determined, provided that the primitives representing the interior and the boundary (if any) can be identified.

Finally, it can be noted that only two out of the three result options defined by ISO 19107 are implemented for the 9-Intersection relationships. Intersection results can be TRUE or FALSE,

but all intersection components are evaluated every time a relationship query is run. Adding the NULL (not evaluated) option would result in a total of 3^9 potential relationships (19,683), which would add an additional level of complexity to the process of mapping user-terminology to relationships. The NULL option is required to filter for objects that match a particular pattern but requires an end-user to have a good understanding of the 9-Intersection matrix. If required can be added to the algorithm at a later date (see 2.4.1 for an explanation of the impact of the NULL option).

7.5 *Finding Intersecting Objects and Find Objects with Relationship*

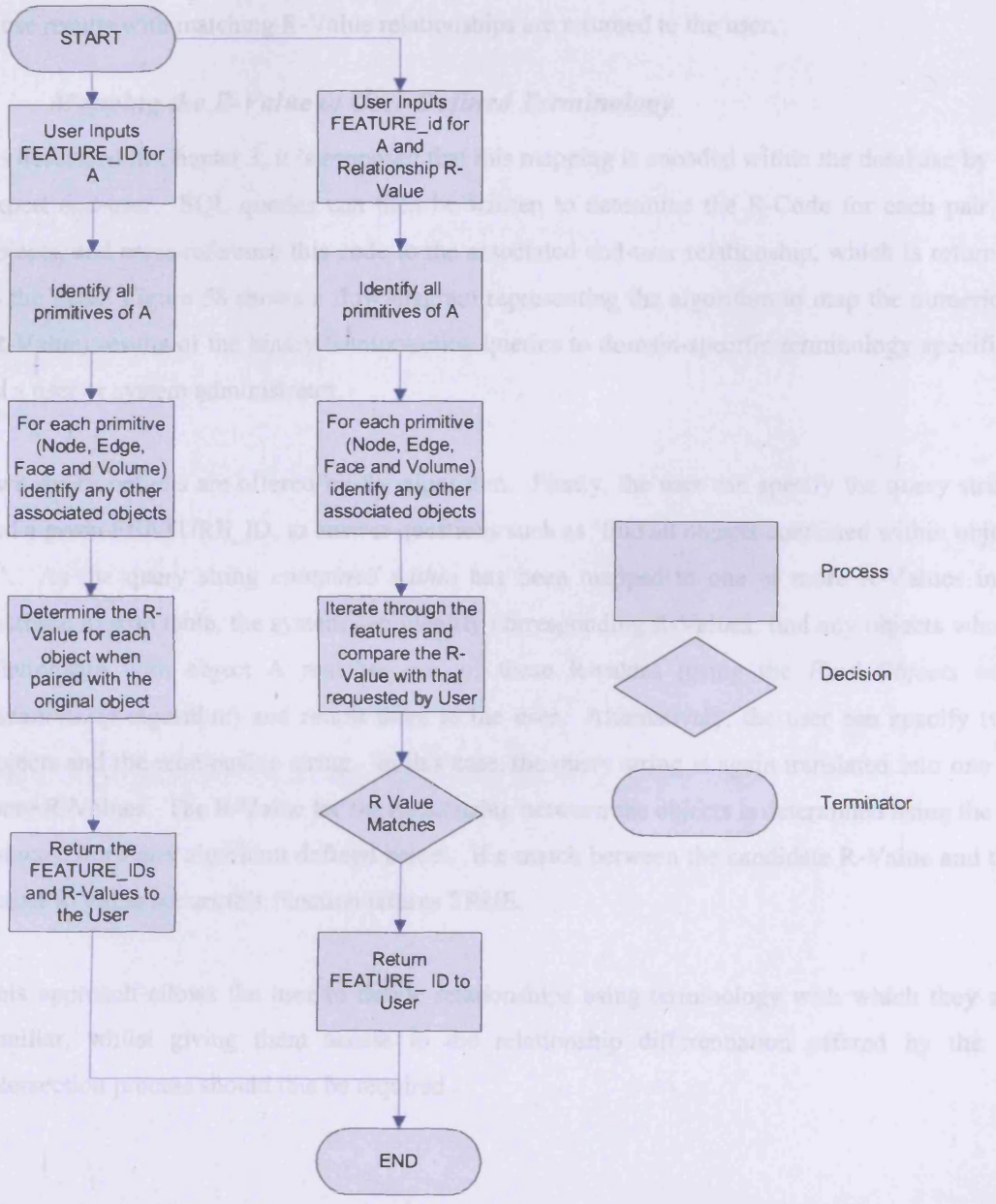


Figure 57 - Flowchart for Find Intersecting Objects, Find Objects with Relationship

Figure 57 details the algorithm required to identify objects intersecting with a given object. The user is offered two options. If the user specifies a FEATURE_ID for A, then the algorithm first identifies any primitives associated with A. For each primitive, it then identifies any other associated objects, grouping the results. The R-Value of the 9-Intersection relationship between object A and all other identified objects is calculated, and the result (a list of intersecting objects and the specific R-Value for the intersection) returned to the user.

If the user specifies a FEATURE_ID for A along with a specific relationship R-Value, the process described above is repeated – i.e. all intersecting objects are identified. However, only those results with matching R-Value relationships are returned to the user.

7.6 Mapping the R-Value to User-Defined Terminology

As described in Chapter 3, it is proposed that this mapping is encoded within the database by an expert end-user. SQL queries can then be written to determine the R-Code for each pair of objects, and cross-reference this code to the associated end-user relationship, which is returned to the user. Figure 58 shows a flow diagram representing the algorithm to map the numerical (R-Value) results of the binary 9-Intersection queries to domain-specific terminology specified by a user or system administrator.

Two query options are offered by the algorithm. Firstly, the user can specify the query string and a given FEATURE_ID, to answer questions such as 'find all objects contained within object A'. As the query string *contained within* has been mapped to one or more R-Values in a database lookup table, the system can identify corresponding R-Values, find any objects whose relationship with object A matches one of these R-values (using the *Find Objects with Relationship* algorithm) and return them to the user. Alternatively, the user can specify two objects and the relationship string. In this case, the query string is again translated into one or more R-Values. The R-Value for the relationship between the objects is determined using the *9-Intersection Pairs* algorithm defined below. If a match between the candidate R-Value and the actual R-Value occurs this function returns TRUE.

This approach allows the user to define relationships using terminology with which they are familiar, whilst giving them access to the relationship differentiation offered by the 9-Intersection process should this be required.

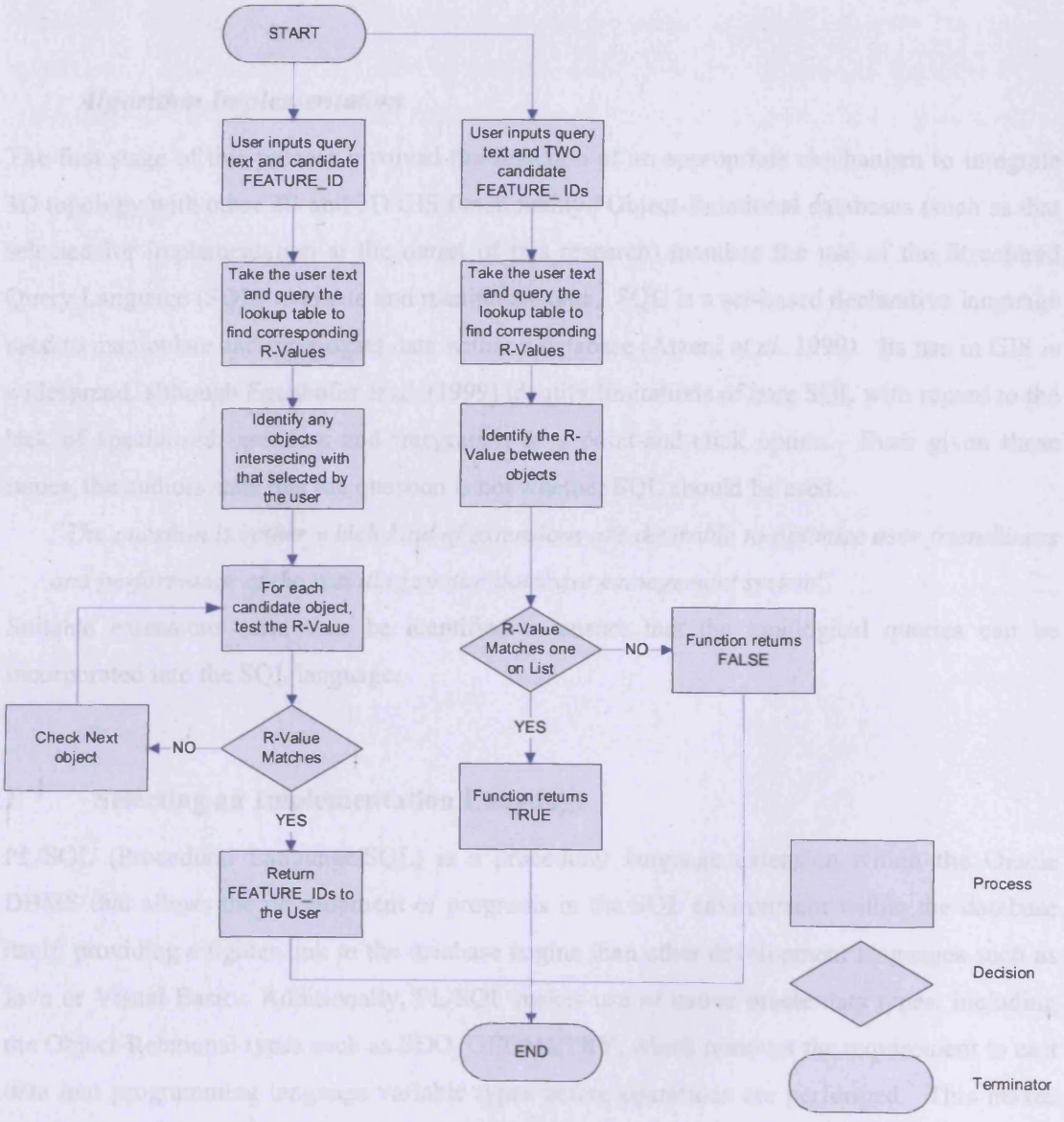


Figure 58 - Algorithm for Mapping User Defined Terminology to R-Values

7.7 *Algorithm Implementation*

The first stage of this process involved the selection of an appropriate mechanism to integrate 3D topology with other 2D and 3D GIS functionality. Object-Relational databases (such as that selected for implementation at the outset of this research) mandate the use of the Structured Query Language (SQL) to create and manipulate data. SQL is a set-based declarative language used to manipulate and interrogate data within a database (Atzeni *et al.* 1999). Its use in GIS is widespread, although Egenhofer *et al.* (1999) identify limitations of core SQL with regard to the lack of specialised operators and integration of a point-and-click option. Even given these issues, the authors state that the question is not whether SQL should be used:

> *"The question is rather which kind of extensions are desirable to optimize user friendliness and performance of the resulting spatial database management system".*

Suitable extensions must thus be identified to ensure that the topological queries can be incorporated into the SQL language.

7.7.1 Selecting an Implementation Language

PL/SQL (Procedural Language/SQL) is a procedural language extension within the Oracle DBMS that allows the development of programs in the SQL environment within the database itself, providing a tighter link to the database engine than other development languages such as Java or Visual Basic. Additionally, PL/SQL makes use of native oracle data types, including the Object-Relational types such as SDO_GEOMETRY, which removes the requirement to cast data into programming language variable types before operations are performed. This makes PL/SQL an ideal programming language where intensive SQL querying is required. Given that this is the case here, PL/SQL has been selected over other available options (such as C++ or Java) for implementation[11].

PL/SQL allows standard programming constructs such as variables and loops to be added to the set-based SQL queries. It can be used to wrap the complex, multi-stage 9-Intersection algorithms into more simple statements. Although users could directly execute SQL queries required to identify shared primitives, the complex statements required would be beyond many users. Users would also have to evaluate each component of the 9-*Intersection* matrix separately and then calculate the R-Value according to the formula given in Chapter 3.

[11] Note that this choice has been made due to the predominance of SQL queries in the code and the simplicity of the algorithms. Should more complex algorithms be required, the advantages in terms of performance of compiled C++ will outweigh the use of PL/SQL.

Extending SQL in this manner results in an interoperable approach whereby topological functionality was is made available to multiple GIS packages. It permits topology to be integrated with non-spatial queries, directional queries and metric queries, presenting a single query interface to the GIS developer or end-user.

A review of two implementations of 2D and 2.5D topology in an Object-Relational database can be identified – Oracle (Oracle 2007b) and 1Spatial's Radius Topology (Laser-Scan 2007) – validated this approach. In both cases, procedural extensions to SQL were used to wrap topological functionality to simplify the required SQL for developers and specialist end-users. Developers in turn can add further wrappers to present a graphical user interface to the queries.

Further information relating to databases, SQL, existing implementations of topology in Object-Relational databases, options for extending SQL, the PL/SQL language and the implementation environment can be found in Appendix 8.

7.7.2 Implemented Packages

A number of PL/SQL packages were created to implement the 9-Intersection algorithms against each data structure (STS and 3DFDS). In general, functions developed for both STS and 3DFDS are identical. The main difference occurs in relation to the process of identifying interior and boundary primitives for each object, which for 3DFDS involves following a number of relational joins and querying appropriate exception tables. In order to facilitate code reuse, code was implemented against the Extended 3DFDS structure to return PL/SQL tables emulating the structure provided by the TOPO_ tables in STS. Packages are summarised in Table 54 below.

Package Name	*Description*
TOPOLOGY	Provides mapping from user terminology to relationship R-Values.
WHAT_RELATIONSHIP	Identifies objects having a specific relationship with object A, and objects intersecting with object A.
NINE_INTERSECTIONS	Calculates the R-Value for the 9-Intersection relationship between two objects. Also provides functionality to determine the R-Value between two part objects, and to query the individual components of the 9-Intersection relationship.
SHARE_NODE	Identifies any shared Node primitives between two objects. In 3DFDS, this function also contains code to construct the TOPO_NODE table for a particular object.
SHARE_EDGE	Identifies any shared Edge primitives between two objects. In 3DFDS, this function also contains code to construct the TOPO_EDGE table for a particular object.
SHARE_FACE	Identifies any shared Face primitives between two objects. In 3DFDS, this function also contains code to construct the TOPO_FACE table for a particular object.
SHARE_VOLUME	Identifies any shared Volume primitives between two objects. In 3DFDS, this function also contains code to construct the TOPO_VOLUME table for a particular object.
SHARED_ROUTINES_9I	Generic functionality to determine object dimension and to determine whether it is possible to reconstruct an object from the primitives shared with another object.

Table 54 - PL/SQL Packages

Appendix 6 contains details of the functions developed.

7.8 Determining the Relationships using SQL

A number of examples are given below to illustrate the calls made from SQL to utilise the PL/SQL routines developed. These are presented as a series of questions that a typical end-user may ask of the database. DUAL is the name of the dummy table used by Oracle when a query does not logically have a table name. Queries referencing PART FEATURE IDs are those where the relationship between part of each object is being compared, rather than the whole object.

7.8.1 Relationship Determination Using R-Code Values

Question: What is the R-Value of the relationship between Object 1 and Object 2?

```
SELECT NINE_INTERSECTIONS.GET_GEOM_R_VALUE((<<FEATURE_ID1>>,<<FEATURE_ID2>>) FROM DUAL;
```

Question: What is the R-Value of the relationship between Part Object 1 and Part Object 2?

```
SELECT NINE_INTERSECTIONS.GET_R_VALUE((<<PART FEATURE_ID1>>,<<PART FEATURE_ID2>>) FROM
DUAL;
```

Question: Does the Boundary of Part Object 1 intersect the Boundary of Part Object 2?

```
SELECT NINE_INTERSECTIONS.GET_FIELD_1(<<FEATURE_ID1>>,<<FEATURE_ID2>>) FROM DUAL;
```

Question: Which objects have relationship R131 with Object 1?

```
SELECT * FROM
TABLE(WHAT_RELATIONSHIP.FIND_OBJECT_WITH_RELATIONSHIP(<<FEATURE_ID1>>,131));
```

Question: Which objects intersect Object 1?

```
SELECT * FROM TABLE(WHAT_RELATIONSHIP.FIND_INTERSECTING_OBJECTS(<<FEATURE_ID1>>));
```

7.8.2 Relationship Determination Using User Terminology

Question: What artefacts are contained within this site?

```
SELECT * FROM TABLE(TOPOLOGY.QUERY('CONTAINED',<< FEATURE_ID>>));
```

Question: Which buildings are impacted by this road extension?

```
SELECT * FROM TABLE(TOPOLOGY.QUERY('IMPACTED',<< FEATURE_ID>>));
```

Question: Which buildings are next to this road?

```
SELECT * FROM TABLE(TOPOLOGY.QUERY('NEXT TO', << FEATURE_ID>>));
```

Question: Is this geological fault inside this rock formation?

```
SELECT TOPOLOGY.QUERY_PAIR(<< FEATURE 1>>, << FEATURE 2>>) FROM DUAL;
```

Question: Which objects have some form of topological relationship with this one?

```
SELECT * FROM TABLE(TOPOLOGY.QUERY_ANY(<< FEATURE_ID>>));
```

7.9 Reviewing Implementation Quality

Optimal execution of queries against all three structures is required to ensure a fair basis for comparison. The code quality review processes followed to validate the SQL queries embedded in the PL/SQL code are described here. Sample results of this process are also presented for illustrative purposes.

The SQL in each PL/SQL package was extracted and each query examined individually to validate optimal execution. To facilitate this process an application development tool written by Quest Software (TOAD 2006) and known as "TOAD for Oracle" was used. This tool provides a user-friendly graphical interface to analyse SQL statements[12].

7.9.1 SQL Quality Review

Oracle (2006e) summarise the main goals of SQL tuning process as: the reduction of workload – for example, by means of indexes if a query only needs to query the small percentage of the dataset; balancing the workload – for example by scheduling jobs when no users are connected to the system and parallelizing the workload; ensuring that the execution plan selected by the database (specifically, the query optimisation process) is as efficient as possible.

An execution plan describes the order in which aspects of each query are run – which table in a join is queried first, which join algorithm is used and which indexes are used to improve query performance. In general, tuning SQL by examining an execution plan (known as an Explain Plan within the Oracle context) allows a developer to optimise query performance by ensuring that SQL queries are not using full table scans (reading all the data in a table) to determine the required result[13]. Reviewing the proposed execution plan thus allows developers to suggest alternative plans to the optimiser as the plan generated by the database may be overruled by means of hints, which suggest the use of various indexes or join types to the optimiser. Further details relating to executing queries within an Oracle environment can be found in Appendix 7. Table 109 in this Appendix gives the structure of an Explain Plan output.

The first phase of the SQL review process involved evaluating and comparing a number of approaches to the implementation of the INTERSECT operation described in the logical algorithms. Three approaches were identified and Explain Plan statements generated for simple queries representing each approach. The results were used to ensure that the most efficient implementation of the INTERSECT operation was selected.

This was followed by a three step process carried out for each SQL query embedded in the PL/SQL procedures. The original query was executed without any hints and an Explain Plan generated by the Oracle database. This was then reviewed, and areas of concern such as full table scans were identified. Where appropriate an alternative query was written, including hints to the optimiser to make use of specific indexes, alternative join orders or specific join types, the

[12] Further details relating to TOAD software can be found in Appendix 7.

[13] Note that this rule may not be relevant where table sizes are sufficiently small.

query was re-executed and a new Explain Plan generated and reviewed. Execution times for the original and revised queries were also compared.

To account for changes in Explain Plan output due to varying dataset sizes, the queries were executed against a dataset containing 135,168 objects. This was selected as a compromise as it represents one eighth of the maximum dataset size (1.08 million records) and 512 times the minimum dataset size. TOAD (TOAD 2006) software was used to facilitate the generation of Explain Plans, although this process can also be carried out within the Oracle SQL Plus interface.

7.9.2 Selecting a Join Approach

A summary of the three INTERSECT approaches and their Explain Plan results is given in Table 55. Although algorithm time complexity (which measures behaviour of the algorithm as the volume of input data increases) for all three operations is identical, there are two sort operations and an intersection operation for the INTERSECT operator, making this less efficient than either the INNER JOIN or the IN approach. Additionally the use of the IN statement is only advisable when the list of comparisons is small. As can be seen, the Oracle Optimisation process has translated the IN query to a HASH JOIN query for efficiency and that this statement is equivalent to the INNER JOIN. Thus it can be concluded that the INNER JOIN approach provides the most efficient implementation for the required INTERSECT operations.

Approach	*Explain Plan Result*	*Time Complexity*
Using Oracle's INTERSECT operator	SELECT STATEMENT INTERSECTION SORT INDEX SORT INDEX	O(nlogn)
Using an INNER JOIN	SELECT STATEMENT HASH JOIN INDEX INDEX	O(nlogn)
Using an IN statement	SELECT STATEMENT HASH JOIN INDEX INDEX	O(nlogn)

Table 55 – Approaches to implementation of set INTERSECTION

In the case of the algorithms described in this Chapter (which require multiple join queries), and using the dataset described in Chapter 6 (which returns a low number of records for each join) it is likely that an index scan will be more efficient than a full table scan in all cases, as there are no situations where a query returns large volumes of data to the application. The use of nested loop join, as opposed to the hash joins suggested above, may also be appropriate in some cases, as each side of some joins returns very few records (a hash join is optimised to handle larger datasets). Join operations are described more fully in Appendix 7.

7.9.3 SQL Tuning

All full table scans were eliminated through the review process described. Individual query tuning also allowed the identification of cases where hints (specifying that a particular index should be used) improved query performance, and conversely situations where the use of hints to remove full table scans reduced query performance. An example of the results obtained are given here, along with a comparison of the INTERSECT and INNER JOIN operator. A full set of results can be found on the attached CD.

Example 1 – Using Hints to Improve Query Performance

The following query determines the ID of any Part Objects with Face primitives containing Node Exceptions that are shared with the Nodes associated with FEATURE_ID = 11. Both versions of the query shown return identical results.

```
SELECT PARENT_TOPO_ID FROM TOPO_FACE GGF INNER JOIN
(SELECT FACE_ID FROM NODE_FACE_EX GNFE INNER JOIN
(SELECT NODE_ID FROM TABLE (GEN_SHARE_NODE.GET_ALL_NODES (11))) B
ON GNFE.NODE_ID = B.NODE_ID) X
ON GGF.FACE_ID = X.FACE_ID
```

Operation	*Object Name*	*Rows*	*Bytes*	*Cost*
SELECT STATEMENT Optimizer Mode=ALL_ROWS		16 K		94.76377
HASH JOIN		16 K	322 K	94.76377
HASH JOIN		16 K	175 K	45.60327
COLLECTION ITERATOR PICKLER FETCH	.GET_ALL_NODES			
TABLE ACCESS FULL	. NODE_FACE_EX	18 K	166 K	9.29244
TABLE ACCESS FULL	. TOPO_FACE	129 K	1 M	46.97887

Table 56 – Explain Plan Results for Part Object Query

Table 56 above shows the Explain Plan output for the execution of the above query against the Enhanced 3DFDS data structure. As can be seen, two full table scans are involved. Although the NODE_FACE_EXCEPTION table is relatively small in size, it is possible that the storage required for the TOPO_FACE table could be significant. Therefore an alternative query, avoiding the full table scans, is required. This is shown below.

```
SELECT /*+ USE_NL (GGF X)*/ PARENT_TOPO_ID FROM GEN_GEOM_FACE GGF INNER JOIN
(SELECT /*+ USE_NL (GNFE B) */ FACE_ID FROM GEN_NODE_FACE_EX GNFE INNER JOIN
(SELECT NODE_ID FROM TABLE (GEN_SHARE_NODE.GET_ALL_NODES (11))) B
ON GNFE.NODE_ID = B.NODE_ID) X
ON GGF.FACE_ID = X.FACE_ID
```

Operation	*Object Name*	*Rows*	*Bytes*	*Cost*
SELECT STATEMENT Optimizer Mode=ALL_ROWS		16 K		*49187.61805*
TABLE ACCESS BY INDEX ROWID	.TOPO_FACE	1	9	2.00322
NESTED LOOPS		16 K	322 K	49187.61805
NESTED LOOPS		16 K	175 K	16414.91234
COLLECTION ITERATOR PICKLER FETCH	.GET_ALL_NODES			
INDEX RANGE SCAN	. NODE_FACE_EX_PK	1	9	1.00120
INDEX RANGE SCAN	.TOPO_FACE_IDX	1		1.00214

Table 57 – Revised Explain Plan Results for Part Object Query

Table 57 shows the Explain Plan output for the modified query. As can be seen, the hints written into the query suggested the use of a NESTED LOOP join as opposed to the HASH join suggested by Oracle's optimiser. A NESTED LOOP was considered appropriate here due to the low number of records returned by each side of the join query. The NESTED LOOP join in turn substituted the full table scans with index range scans. The original query gave a performance range from 125 milliseconds on initial execution, down to 109 seconds after 10 repetitions. The modified query ran for 109 milliseconds on first execution, reducing this to 62 milliseconds following 10 repetitions – a saving of approximately 50% of execution time[14].

The above query forms one part of the FIND_INTERSECTING_OBJECTS routine for the Extended 3DFDS structure. Applying the NESTED LOOP joins across the entire query reduced query time from 8 seconds (2 after 10 repetitions) to 421 milliseconds (156 after 10 repetitions), reducing the execution time to $1/12^{th}$ of that obtained for the original query.

[14] Note that the creation of a correct execution plan in Oracle depends on the availability of accurate database statistics, including table size, index size, distribution of the data in the index and so forth. These statistics were calculated prior to test execution. Further investigation is required to identify whether this was an issue in this situation.

Example 2 – Intersect versus Inner Join

The following query identifies the IDs of the FEATUREs sharing a Node with FEATURE_ID = 1. In the first case, the query to uses an INTERSECT relationship.

```
SELECT DISTINCT F.FEATURE_ID FROM TOPO_PART_TABLE F INNER JOIN
(SELECT PARENT_TOPO_ID FROM TOPO_NODE A INNER JOIN
(SELECT NODE_ID FROM TOPO_NODE WHERE PARENT_TOPO_ID = 1
INTERSECT
SELECT NODE_ID
FROM TOPO_NODE WHERE PARENT_TOPO_ID != 1) C
ON A.NODE_ID = C.NODE_ID) D
ON F.PARENT_TOPO_ID = D.PARENT_TOPO_ID
WHERE F.FEATURE_ID != 1
```

Operation	*Object Name*	*Rows*	*Bytes*	*Cost*
SELECT STATEMENT Optimizer Mode=ALL_ROWS		1		2686.74577
SORT UNIQUE		1	32	2686.74577
NESTED LOOPS		10	320	2685.74561
NESTED LOOPS		9	207	2676.74468
VIEW		7	91	2662.72184
INTERSECTION				
SORT UNIQUE		7	70	
INDEX RANGE SCAN	ZLATANOVA_512.GEOM_NODE_BND_IDX	7	70	3.00333
SORT UNIQUE		881 K	8 M	
TABLE ACCESS FULL	ZLATANOVA_512.GEOM_NODE	881 K	8 M	382.26041
TABLE ACCESS BY INDEX ROWID	ZLATANOVA_512.GEOM_NODE	1	10	2.00326
INDEX RANGE SCAN	ZLATANOVA_512.GEOM_NODE_NODE_IDX	1		1.00216
TABLE ACCESS BY INDEX ROWID	ZLATANOVA_512.TOPO_PART_TABLE	1	9	1.00226

Table 58 – Explain Plan Results for Objects Sharing Node Query, INTERSECT operator

As can be seen from Table 58, the resulting execution plan includes a FULL TABLE SCAN. An alternative version of this query can also be implemented, as shown here.

```
SELECT DISTINCT E.FEATURE_ID FROM TOPO_PART_TABLE E INNER JOIN
 (SELECT D.PARENT_TOPO_ID FROM TOPO_NODE D INNER JOIN
 (SELECT A.NODE_ID FROM TOPO_NODE A INNER JOIN TOPO_PART_TABLE B
 ON A.PARENT_TOPO_ID = B.PARENT_TOPO_ID WHERE
 B.FEATURE_ID = 1) C ON D.NODE_ID = C.NODE_ID) F
 ON F.PARENT_TOPO_ID = E.PARENT_TOPO_ID WHERE E.FEATURE_ID != 1
```

Operation	*Object Name*	*Rows*	*Bytes*	*Cost*
SELECT STATEMENT Optimizer Mode=ALL_ROWS		9		28.02813
SORT UNIQUE		9	342	28.02813
NESTED LOOPS		9	342	27.02800
NESTED LOOPS		9	261	18.02706
NESTED LOOPS		7	133	4.00451
INDEX RANGE SCAN	ZLATANOVA_512.TOPO_PART_TBL_IDX	1	9	2.00214
INDEX RANGE SCAN	ZLATANOVA_512.GEOM_NODE_BND_IDX	7	70	2.00237
TABLE ACCESS BY INDEX ROWID	ZLATANOVA_512.GEOM_NODE	1	10	2.00322
INDEX RANGE SCAN	ZLATANOVA_512.GEOM_NODE_NODE_IDX	1		1.00214

Table 59 – Revised Explain Plan Results for Objects Sharing Node Query, INNER JOIN

Table 59 shows the Explain Plan output for the second query. The first query involves a full table scan, and gives and execution time of 1.32 seconds, whereas the second query took 30 milliseconds to run.

7.9.4 Reviewing the Tuning Process

The query hints resulting from this process were optimised for the 135,168 Object dataset and may in fact reduce performance for other dataset sizes. Even if the schemas are the same, the optimizer can choose different execution plans if the costs are different. Therefore the hints embedded in the PL/SQL code may, in some cases, result in reduced rather than improved performance.

SQL query tuning was carried out on a query-by-query basis, running each query separately. As described above, Operation values from the Explain Plan output and execution time were taken

into account when identifying an optimal execution plan. However, this process does not take into account any performance gains to be made through caching of data in memory, or through the pinning of queries due to their repeated execution and the use of bind variables.

Although NESTED LOOP joins are the least efficient of the join types, their use was considered appropriate for a number of join queries due to the low number of records expected from each side of the join query, which is in turn a factor of the selected dataset (see Section 9.12.3 for a full analysis of the impact of the artificial dataset). Thus although the HASH JOIN suggested by the optimiser may be appropriate due to the relatively large number of records in the table, this suggestion does not take the nature of the expected result in to account. Again, performance, and hence the appropriate hint, may differ for more complex objects or for objects having many parts.

The use of hints relating to index use and join order improved the performance of various queries. However, embedding these hints in the PL/SQL code may result in poor performance when the code is run against datasets with differing characteristics in terms of size or more complex object types.

7.10 Summary

This Chapter described algorithms developed for the identification of 9-Intersection Relationships between objects using the three structures (As-Required, STS and Extended 3DFDS). In the absence of a topology engine, a Proxy for the As-Required queries was identified utilising existing Oracle query functionality. Structure-based algorithms were then described in terms of identifying relationships between simply connected part objects, with additional processes being presented to handle the case of multi-part compound objects. A brief overview of the implementation of these algorithms in the context of an Oracle database was given, and a number of sample queries presented to illustrate how the PL/SQL procedures developed can be incorporated into standard SQL queries.

Although PL/SQL code was re-used where possible, a number of implementation differences between STS and 3DFDS relating to the identification of boundary and interior primitives were presented. These related to the requirement in 3DFDS to derive this information from the relationships between primitives.

In order to optimise performance, the SQL embedded in the PL/SQL algorithms was extracted and execution plans for each query generated. These were reviewed and hints embedded in the queries where required. The implementation has been described here in the context of the

selected database (Oracle 10g, Oracle 2006). However, it can be noted that the algorithms described here can be implemented within any database supporting Object-Relational spatial data types and procedural extensions to SQL.

Taking the dataset described in Chapter 6 and these algorithms, Chapter 8 describes a series of tests to compare the three structures – the Proxy for As-Required calculations, STS and 3DFDS.

8 Test Design

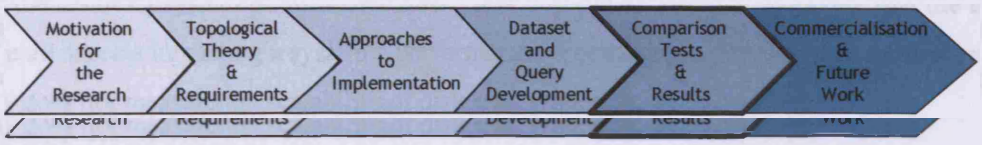


Figure 59 - Overview of Document Structure showing Context of this Chapter

8.1 Introduction

Bringing together the data structures from Chapters 4 and 5, the test dataset described in Chapter 6 and the query algorithms described in Chapter 7, this Chapter describes the performance tests used to compare the As-Required, STS and Extended 3DFDS data structures. An overview of concepts considered in the context of computer system testing is first presented, followed by a brief description of the tests themselves and of the test execution process. The Chapter concludes with a discussion predicting expected test outcomes.

8.2 Scalability

Allamaraju *et al.* (2000) describe performance as the measure of end-to-end response time of a system requests from a single user at a time. Scalability builds on this concept, and is the ability to maintain response times as the number of simultaneous users increases. Ye *et al.* (2002) further this definition:

> *"A system is scalable if there is a straightforward way to upgrade it to handle an increase in traffic while maintaining adequate performance."*

This implies that no software or architectural changes are required and the system can be scaled simply by adding more powerful hardware.

However, beyond a certain point the introduction of additional hardware does not necessarily resolve issues (Oracle 2007d). In fact, one of the main aims of scalability testing is to identify potential system bottlenecks (network, memory, disk and processor contention) that will cause issues as the number of concurrent users increases. In an ideal world, response times for one hundred concurrent users would be equal to those for a single user. However, in practice linear growth is often seen as simultaneous users increase, and a non-scalable system may rapidly reach a point of exponential growth.

Understanding system architecture is also fundamental to scalability prediction. For example, the PL/SQL code implementing the binary queries forming part of this research is executed entirely within the database server environment, leading to potential resource contention issues on this single machine.

Scalability can be measured by plotting a graph of increasing number of simultaneous users against the resulting system performance time. It is important, however, to ensure that the tests measure all aspects impacting a system's performance. For example, running tests against small datasets does not measure the scalability of disk read processes.

8.3 Algorithm Time Complexity

Along with code quality and implementation hardware, this is an important factor that determines the overall expected running time of a program (Aho *et al.* 1987, pg. 16). Understanding the time complexity of an algorithm provides an insight into how the algorithm will perform as the size of input grows. Size of input here refers to number of elements or records forming the input of a particular function, query or sub-query, rather than the physical size of the data in question. Worst-case time complexity is described using "big oh" notation. For example O(n) (Order(n)) denotes a function whose execution time grows linearly with the number of inputs. Ideally, time complexity will be linear or better, although this is not always possible. In general, when determining the order (or time complexity) of a particular function the efficiency is expressed as the term with the highest order or a combination of such terms.

Although the algorithms encoded in the procedural element of the PL/SQL code written as part of this research are relatively simple, the use of SQL queries hides the complexity of the underlying operations, including joins, set operations and index searches. Table 60 (partly sourced from Libkin 2005) provides an overview of time complexity metrics for common relational operations encountered in SQL queries. As can be seen, the value that n (size of input) represents may vary. For example, in join operations, n represents the number of records passed into the join on either side. The Value column in Table 60 gives the interpretation of n in each case.

Table 60 - Algorithm Time Complexity of Relational Operations (Libkin 2005)

The above values (Table 60), along with the Explain Plan execution plan results captured as part of the SQL quality review process, can be used to evaluate the time complexity of the *9-Intersection Pairs* and the *Find Intersecting Objects* and *Find Objects with Relationship* functionality for both the STS and 3DFDS approaches.

8.4 Preliminary Performance Tests

A number of preliminary tests were executed to help to ensure that the main tests emulated a real-world situation as closely as possible, despite the artificial nature of the test dataset and shared test environment (Preliminary Tests 1 and 2) and to optimise the Proxy for the As-Required queries (Preliminary Tests 3 and 4). These tests were designed to ensure that, although the main aim of the testing process was comparison between structures, the query times obtained for the individual structures could also be considered close to those obtainable in a real-world setting.

8.4.1 Preliminary Test 1 – Random or Sequential FEATURE_ID Generation

For the *9-Intersection Pairs* testing processes, tests were carried out using sequentially generated FEATURE_ID values (where the first FEATURE_ID is randomly generated, but the second FEATURE_ID in the pair is determined by adding 1 to the first FEATURE_ID). Sequential FEATURE_ID querying has the advantage of guaranteeing a non-disjoint relationship between objects. For example, relationship R511 (Body/Surface) is modelled by FEATURE_ID 3 and FEATURE_ID 4. Relationship R207 (Line/Line) is modelled by FEATURE_ID 207 and FEATURE_ID 208. The aim of this test was to identify the impact of selecting FEATURE_IDs sequentially – i.e. identifying and querying only objects with non-disjoint topological relationships. This was then contrasted with a random FEATURE_ID generation process. The use of randomly generated FEATURE_IDs ensured that artificial performance gains are not made due to data or index caching (see Section 9.3 for the results of this test).

8.4.2 Preliminary Test 2 – Number of Test Iterations

All tests were executed for iterations ranging from 10 to 10,000. This allowed the investigation of the impact of pinning queries, and their corresponding execution plans, in system memory. Again, the aim of this test was to ensure that the main tests emulated a real-world environment as closely as possible. In this case, it can be assumed that, in a multi-user setting, queries would be pinned into memory and pre-parsed (see Section 9.4 for the results of this test).

8.4.3 Preliminary Test 3 – Identifying Optimal R-Tree Index Tolerance

A total of six different tables, each linked to spatial indexes having differing tolerance values, were created for the As-Required data structure for each dataset size (264 to 1.08 million records). *9-Intersection Pairs* and *Find Intersecting Objects* queries were run against these tables to identify an optimal R-Tree index tolerance value in terms of query performance (see Section 9.5 for the results of this test).

8.4.4 Preliminary Test 4 – Improving the Proxy for As-Required Query

As described in Chapter 7, the current implementation of Oracle offers a 3D filtering query using the R-Tree index (SDO_FILTER). However, this does not take into account the additional time overhead required to determine the exact 9-Intersection relationship between the selected object pairs (using the SDO_RELATE query). To further improve the quality of the results obtained from the Proxy tests, additional tests to identify this overhead in 2D were thus included. The results in 2D can be added to those obtained for the SDO_FILTER in 3D to provide an overall Proxy closer to that which would be obtained once the full queries are implemented in 3D.

An initial attempt was made to create a 2D index on the existing 3D test dataset. However, queries against this index returned 'UNKNOWN MASK' for the relationships between the objects. Due to the replication process, it was also not possible to simply project the 3D data into 2D, as vertical replication would result in repeated objects having identical 2D coordinates. Therefore, for this purpose, a 2D dataset (Ordnance Survey MasterMap™, Ordnance Survey 2006, consisting of 10.38 million records providing topographic mapping for the London area) was used. The nature of this dataset is different to that of the test data (in that it provides continuous coverage rather than isolated pairs of objects). Table 61 and Table 62 detail the process followed to mitigate this issue. To mirror the queries executed in 3D, SDO_FILTER and SDO_RELATE queries (see Chapter 7) were executed to provide the Proxy value as shown in Table 61.

Queries Executed	*Description*	*Result used as Proxy For*
SDO_FILTER	Time required for the system to query the R-Tree index and identify candidate objects.	Result of this test subtracted from all other queries to identify time spent on computational geometry element of each query.
SDO_RELATE with 'anyinteract' parameter	The 'anyinteract' parameter returns TRUE if the two objects are non-disjoint. Subtracting the SDO_FILTER time gives the time required to identify the presence of the non-disjoint relationship.	*Find Intersecting Objects*
SDO_RELATE with 'touch' parameter	The 'touch' parameter returns TRUE if two objects have a TOUCH relationship. Given that the selected dataset represents topographic mapping data for the London area, this relationship is true for all intersecting objects in the dataset.	*Find Objects With Relationship* To take account of the multiple connectivity of the dataset, the result obtained will be divided by the average number of touching objects (determined by querying the dataset).
SDO_RELATE with the 'determine' parameter	The use of 'determine' returns the name of the relationship between the two specified objects.	*9-Intersection Pairs*

Table 61 - 2D Spatial Queries in Oracle

The Proxy can be further improved by taking into account the impact of 3D data on the relationship determination algorithms, assuming that the algorithm to determine the distance of a point from a polygon (given in Chapter 4) is used for relationship determination (as suggested by Nguyen *et al.* 2005). Assuming simple objects, an object in 3D will have 6 Faces, each with 4 Nodes, and an object in 2D will have 1 Face with 4 Nodes. The 2D relationship determination process for two objects thus executes a point-to-polygon distance algorithm 8 times (distance of 4 Nodes from object A to 1 Face of object B) * 2 objects. In 3D, this total is 96 (distance of 8 Nodes from object A to 6 Faces of object B) * 2 objects). Thus a 3D query will require 12 (96/8) times the number of operations. This factor is again divided by 2.

Multiplication factors (shown in Table 62) can then be calculated to bring the Proxy closer that expected for a 3D dataset (see Section 9.6 for the results of these tests).

Query Type	*Comment*	*3D Multiplication Factor*
SDO_RELATE with 'anyinteract' parameter	'anyinteract' returns TRUE as soon as one intersection is identified, and does not execute the remainder of the point-to-polygon distance algorithms. The number of operations required to do this in 2D is identical to that in 3D.	1
SDO_RELATE with 'touch' parameter	Each object in the MasterMap dataset has a TOUCH relationship with an average of 6 others. Thus the result obtained represents a total of 6 TOUCH queries, and should first be divided by 6. It is then multiplied by 6 to approximate the 3D case.	1 (6/6)
SDO_RELATE with the 'determine' parameter	In this case only two objects are queried in both the 2D and 3D cases. The result is multiplied by 6 to approximate the 3D case.	6

Table 62 – 3D Multiplication Factors for As-Required Proxy

Note that worst-case algorithm complexity of both 2D and 3D relationship determination algorithms ($O(n^2)$) has been used to simplify the calculation of the multiplication factor. In fact, as has been seen in Section 4.3.2 the order of complexity in 3D may be as low as O(n log n) (for example Muller and Preparata, 1978, cited in Preparata and Shamos 1985) and O(log n) (Konidaris *et al.*, 2003) for 2D. The impact of this on the results analysis process is discussed in Section 9.12.

8.5 Main Performance Tests

For the main test, groups of 1, 2, 4, 6 and 8 concurrent-user queries were run for each of the three query types (*9-Intersection Pairs, Find Intersecting Objects, Find Objects with Relationships*) over each of the three data structures (Proxy for As-Required, STS and 3DFDS) and for each of the five dataset sizes (264, 2112, 16896, 135168 and 1.08 million objects). Each test was designed to take into account the results of the preliminary tests in terms of the use of an appropriate Proxy for the As-Required queries, the number of test iterations and the use of random or sequential FEATURE_ID generation.

The performance tests have one primary aim – to determine which, if any, of the three data structures provides the most efficient environment for the determination of 3D binary topological relationships within the context of the 9-Intersection framework. Three sets of results were extracted from the main test to meet this requirement

8.5.1 Main Test 1 – Scalability

The scalability of each approach with respect to increasing numbers of users is measured by examining the results obtained as the number of concurrent users is increased. Results of this test are given in Section 9.8.

8.5.2 Main Test 2- Algorithm Complexity

The time complexity of each approach with respect to increasing volumes of data is measured by examining the results obtained as the dataset size increases. Results of this test are given in Section 9.9.

8.5.3 Main Test 3- Workload Variation

The projected performance of each structure under varying workload is tested by comparing the results obtained for each of the three query types (*9-Intersection Pairs, Find Objects with Relationship* and *Find Intersecting Objects*). Results of this test are given in Section 9.10.

8.5.4 Compensating for the Artificial Dataset and Shared Test Environment

Elements were incorporated into the main test design to ensure that the results reflected those obtainable in a real-world environment. The process of reading data from disk significantly increases query time, tests were configured to ensure that executed queries force a disk read operation where possible, ensuring that performance results were not artificially improved. This involved both the use of high volume datasets (1.08 million records) which could not be stored entirely in memory, and the use of random number generation to identify the FEATURE_ID(s) input into each query, ensuring that different areas of the index and elements of data were required to be read into memory from disk.

Given that the tests were to be run against a database that also supported other users, and over a network carrying other traffic, a total of three sets of each test were run. Average execution time values taken from these sets then provide a more representative overall performance measure.

8.6 Test Execution

8.6.1 Generating a Unique Test ID

The first step in the testing process involved the definition of a GROUP_TEST_ID for each of the three runs of each test. Following this, each of the three main test types (*9-Intersection Pairs*, *Find Intersecting Objects*, and *Find Objects with Relationship*) were also assigned a unique TEST_ID. To completely identify a specific test, the final Test ID comprised the sum of

the sum of the GROUP_TEST_ID and the TEST_ID. This Test ID, in combination with the start time for the test is stored as the primary key in a test results table in the database, allowing specific test results to be queried and analysed.

8.6.2 Running the Tests

A series of PL/SQL test harnesses were written to run the tests defined above. These take parameters including the name of the test, the unique Test ID generated according to the above process and the number of iterations required. The PL/SQL test harnesses generate the required random or sequential FEATURE_IDs for each query, running the actual test and recording the start and end time of each test to the TEST_STATS table in the database.

8.6.3 Recording Test Results

Due to the high number of individual tests, a mechanism was also required to automatically store the results of each test within a TEST_STATS table in the Oracle database itself. This table contains information relating to the overall Test ID, test name, start and end time for execution, and the number of iterations forming part of the test. Although a slight overhead will be incurred in writing results to the test table, this is equal for all three structures and will not impact the comparative process. Additionally, the end-time recorded is measured before the test execution details are written to disk, and thus does not include the time to write the results to disk.

8.7 Predicted Test Results

Information relating to the execution plans for individual SQL queries (as discussed in Chapter 7), the implemented PL/SQL algorithms and knowledge of the hardware environment can be combined to provide preliminary, high level, predictions for the outcome of the performance testing processes. The review following this Chapter should, however, be taken into account when considering this assessment. Predictions for the Proxy for As-Required queries were not possible due to the underlying proprietary code.

8.7.1 Preliminary Test 1 – Random or Sequential FEATURE_ID Generation

Given that objects are numbered sequentially, it can be expected that querying the relationships between sequential object pairs will lead to artificially improved performance test results over those obtained for randomly generated FEATURE_IDs, as only one disk read operation is required data for both objects.

8.7.2 Preliminary Test 2 – Number of Test Iterations

Due to the initial query parsing process whereby Oracle determines an execution plan for the SQL and stores it in memory, a higher execution time will be observed for the single iteration tests than others. Once the execution plan is pinned in memory, average measured performance for individual queries over higher numbers of iterations will converge. Thus performance time per query measured at, for example, 100, 1000 or 10000 iterations will be similar, although this may not be the case for lower iterations – for example, if two iterations are used, the time to determine the query execution plan (which can be lengthy) will be a far more significant component of the average time than for 100 queries, driving the value higher. Table 63 illustrates this situation (using artificial numbers). As can be seen, for higher iterations the average tends closer to the value for the second and subsequent queries, providing a better representation of a real-world situation where it is likely that a query will be pre-parsed.

Time for first query (s)	*Time for second and subsequent queries (s)*	*Total number of queries executed*	*Average (s)*
1	0.01	2	0.505
1	0.01	10	0.109
1	0.01	100	0.0199
1	0.01	500	0.01198

Table 63 - Impact of Query Iterations on Average Results

It is also anticipated that performance for the larger datasets (135,168 objects and 1.08 million objects) will be slower than that for smaller datasets, as the use of random FEATURE_ID generation in the PL/SQL test harness ensures that a number of queries will require at least one disk read operation on the latter, whereas all data can be cached in memory for the former.

8.7.3 Preliminary Test 3 – R-Tree Index Tolerance Value

Due to the larger number of objects on individual R-Tree index nodes, and hence the smaller size of the index, query performance for index tolerance values of 100m and 500m is expected to be faster than that for lower values. The small extent of the data (approximately 20,000m in the X direction and 100m in the Y and Z directions for the largest dataset) also suggests that query performance for the 500m index tolerance value will be optimal. Given the extent of the Y and Z directions, performance between the 100m and 500m indices may converge, as the number of Leaf Nodes will be similar in two of the three dimensions. Although the level of discrimination provided by these tolerances may not be sufficient, as filter queries will return large numbers of objects, which then have to be processed by the computational geometry algorithms, they have been included as they reflect the situation that may occur if a densely populated dataset is indexed to support rapid data retrieval for visualisation.

8.7.4 Preliminary Test 4 – Improving the Proxy for As-Required Queries

The index tolerance for the 2D test dataset is set to 0.01 m, and the dataset itself has extents of 60 km. It can be anticipated that an increase in performance time will be observed between the tests, with the SDO_FILTER test (which simply references an R-Tree index) taking the least time to execute, followed by the 'anyinteract' test (which returns TRUE as soon as some form of interaction is identified). The 'touch' test will take some additional time to execute, as the exact nature of the relationship must be determined here.

8.7.5 Main Test 1 - Scalability

The hardware on which the three data structures and corresponding datasets are implemented consists of a 4-processor machine (see Appendix 8). Thus, where memory is not an issue (i.e.

for the small datasets) it can be predicted that end-to-end query performance will drop at or above the 4 concurrent user stage, as CPU contention will then occur. For the larger-sized datasets, however, the additional possibility of disk read queuing must also be considered. This may result in performance reduction for two concurrent users and over.

8.7.6 Main Test 2 Algorithm Time Complexity

8.7.6.1 *9-Intersection Pairs*

Given the test dataset described in Chapter 6, the *9-Intersection Pairs* relationship determination process will not involve join operations involving large numbers of records, as it is only the primitives associated with each object that form inputs into the join processes. Thus although a number of NESTED_LOOP join operations can be identified in the Explain Plans corresponding to the relationship determination process, the time complexity of these will be a constant as the number of records input into the loops will not change as the number of records in the dataset increases. The objects in the 1.08 million-object dataset are replicas of those in the 264 object dataset. It is thus expected that Join operations will not suffer performance degradation and they can thus be ignored in terms of complexity evaluation. A similar argument applies to the SORT operations, which again will deal with relatively constant numbers of input records as dataset size increases.

The PL/SQL code wrapping the SQL queries iterates through the query result set and pipelines the result out to calling procedures. Thus the complexity of this code is O(1) as the number of resulting records (whether they are primitives, values from the TOPO_ tables or other) will not change as the size of the dataset increases.

Of greater interest here are the index-based operations, as index size (and hence number of index nodes) will increase as the number of records in the underlying dataset increases. This must be taken into account when considering time complexity of the algorithms. A review of the Explain Plans for the queries associated with this functionality for STS identified one type of index query– a RANGE_SCAN. As per Table 60, this has time complexity of O(sqrt(n)), where n is the number of records in the table. For 3DFDS, two types of index query were identified – RANGE_SCAN and INDEX_FAST_FULL_SCAN. The latter has time complexity of O(n). These values thus give the expected time complexity for this functionality.

8.7.6.2 *Find Intersecting Objects and Find Objects with Relationship*

Find Intersecting Objects functionality identifies the Nodes associated with an object, and then identifies any other objects also linked to the resulting Nodes list. The second part of the

process takes the intersecting objects and passes them to the 9-Intersection relationship determination algorithm. The latter has expected time complexity as described above.

As with the *9-Intersection Pairs* PL/SQL code above, in the case of the datasets utilised for testing this has time complexity of O(1). Given the pair wise nature of the dataset, the number of intersecting objects will always be 1, no matter whether the data under consideration forms part of the original 264 records or the replicated datasets.

Unlike the SQL utilised to implement the *9-Intersection Pairs*, SQL underpinning this functionality does not directly select a small number of records relating to the primitives of two given objects. For the known object, primitives are easily identified - again, the most complex operation here is an INDEX_RANGE_SCAN having time complexity O(sqrt(n)). Examining the joins involved in the process of identifying the unknown objects identifies the use of HASH JOINS for STS and NESTED_LOOP joins for 3DFDS. However, these are executed following the index filtering processes – i.e. against a small number of records. For STS, an INDEX_RANGE_SCAN and an INDEX_FULL_SCAN are required to identify any intersecting objects. This results in time complexity of O(n) + O(sqrt(n)). For 3DFDS, INDEX_RANGE_SCAN and INDEX_FULL_SCAN operations are also involved, thus giving the same predicted time complexity. Table 64 below summarises predicted Algorithm Complexity. Note that the values shown represent the worst-case predictions – test results may reveal that the Oracle database is in fact performing far more efficiently.

Structure	*Test*	*Predicted Algorithm Complexity*
STS	*9-Intersection Pairs*	O(sqrt(*n*))
STS	*Find Intersecting Objects*	O(n) + O(sqrt(n))
STS	*Find Objects with Relationship*	O(n) + O(sqrt(n))
3DFDS	*9-Intersection Pairs*	O(n)
3DFDS	*Find Intersecting Objects*	O(n) + O(sqrt(n))
3DFDS	*Find Objects with Relationship*	O(n) + O(sqrt(n))

Table 64 - Predicted Algorithm Complexity

Due to the use of a commercial database (Oracle), time complexity values for the index, sort and join algorithms actually implemented within the database are not available. The actual time complexity may differ widely from that described in Table 64. Time complexity measures do not take into account the requirement for secondary storage (and hence disk read operations) when handling larger datasets (Aho *et al.* 1987). Disk access relates to the transfer of blocks from disk (where a block is the smallest unit of data transfer between disk and memory) and are much slower than direct in-memory access to data. In fact, the time spent on disk access is likely to dominate a query (Silberschatz *et al.* 2002). There is also a need to distinguish

between sequential reads and random disk reads – the former occur when the blocks are adjacent on disk and are faster as the disk does not have to spin to the correct area or to move to the desired track. Appendix 4 describes measures taken during the data creation process to avoid the performance gains due to sequential disk access.

These operations may negatively impact the observed performance as data volumes increase, resulting in performance degradation higher than that predicted.

It has also been assumed that, although joins are generally time-costly operations, they will not impact overall time complexity here as the number of records passed into the join will be small due to the index-based filtering operations. However, particularly for cases where the number of joins to be followed is high (such as the *Find Intersecting Objects* query for 3DFDS) this assumption may be incorrect.

8.7.7 Main Test 3 - Workload Variation

As described in Chapter 7, the algorithms for the *Find Intersecting Objects* and the *Find Objects with Relationship* queries call the *9-Intersection Pairs* algorithms. It is thus expected that the former will perform more slowly than the latter. The *Find Intersecting Objects* and *Find Objects with Relationship* algorithms are identical, with the latter having an additional test for a specific relationship. Given the pair-wise nature of the dataset (and hence the fact that there will only be one intersecting object returned by the query) the difference in performance between these queries will be negligible. For the As-Required structure, similar performance is expected for both valid query types (*9-Intersection Pairs* and *Find Intersecting Objects*) as both these queries first identify the R-Tree index Node for the first object and then either list all other objects on this Node (*Find Intersecting Objects*) or iterate through the list to determine if the second FEATURE_ID is also present.

8.8 Summary

This Chapter described tests designed to compare the three selected data structures (As-Required, STS, 3DFDS). An overview of scalability and algorithm complexity testing was first given, followed by a description of four preliminary tests, utilised to narrow down the range of possible results to be investigated and to compensate for the artificial test environment. These include tests to identify an appropriate index tolerance value for the R-Tree index, and tests to identify improvements to the selected Proxy for As-Required calculations.

Three main test types were identified – scalability (increasing numbers of users), algorithm complexity (increasing dataset size) and workload variation (comparative performance the

different query types). Test outcomes, based on the selected dataset, were predicted, although it was noted that due to implementation in a commercial database, predictions for time complexity of the SQL operations may not be accurate.

The results obtained from the testing processes described here are given in Chapter 9.

9 Performance Test Results

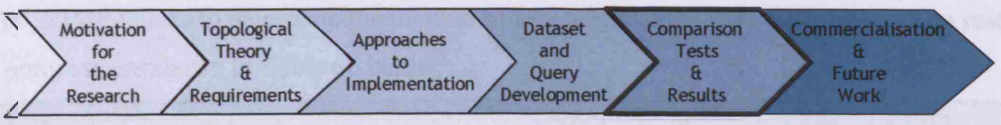


Figure 60 - Overview of Document Structure showing Context of this Chapter

9.1 Introduction

Performance tests implemented as part of this research had one primary aim – to determine which, if any, of the three data structures under consideration provided the most efficient implementation for the determination of binary topological relationships using the 9-Intersection framework. Two main criteria were considered in terms of this comparison – scalability with respect to increasing numbers of users, and time complexity with respect to increasing volume of data. Additionally, each structure was examined in terms of performance under varying query workload.

Logical and physical algorithm and test designs for the performance tests were described in Chapter 7 and Chapter 8. This Chapter presents the results of the comparative performance tests carried out for the three approaches to topological relationship identification, namely As-Required calculation, 3DFDS and STS. Throughout the tests, a standard dataset (described in Chapter 6) was used, replicated to a total of 1.08 million objects with corresponding topological primitives. The three queries identified in Chapter 3 (*9-Intersection Pairs, Find Intersecting Objects* and *Find Objects with Relationship*) were utilised for testing purposes.

9.2 Calculating Test Results from Raw Data

Raw test results take the format shown in Table 65 below.

TEST_NAME	*TEST_ID*	*START_TIME*	*END_TIME*	*ITERATIONS*
9-Interesction Pairs	91000003	05-SEP-06 09.33.12.940000 PM	05-SEP-06 09.33.19.910000 PM	100
9-Interesction Pairs	93000003	05-SEP-06 09.41.15.665000 PM	05-SEP-06 09.41.21.775000 PM	100
9-Interesction Pairs	95000003	05-SEP-06 09.48.34.371000 PM	05-SEP-06 09.48.40.229000 PM	100

Table 65 – Raw Performance Test Results

A total of 3 records are returned for this query – i.e. the test was executed three times. To simplify test identification, each iteration of the test has been assigned a unique Test Group ID (in this case tests starting with 910, 930 and 950 represent the *9-Intersection Pairs* tests using random FEATURE_ID generation).

9.2.1 Extracting Test Execution Times

The first part of the results extraction process involved the subtraction of the END_TIME and START_TIME values to determine the number of seconds taken to execute the test. The results of this process are shown in Table 66 below.

TEST_NAME	*TEST_ID*	*TEST_TIME (s)*	*ITERATIONS*
9-Interesction Pairs	91000003	6.97	100
9-Interesction Pairs	93000003	6.11	100
9-Interesction Pairs	95000003	5.858	100

Table 66 – Results of SQL Query to Determine Overall Test Time

SQL queries were run against the raw results, filtering on the required Test ID and Number of Concurrent Users to group results and determine minimum, maximum and average query execution times. A sample of the results obtained is shown in Table 67.

TEST_NAME	*RANDOM_OR_ SEQUENTIAL*	*NUM_ USERS*	*MIN (s)*	*MAX (s)*	*AVG (s)*	*STDDEV*
9-Intersection Pairs	RANDOM	1	0.05858	0.0697	0.06312	0.00583

Table 67 – Minimum, Maximum, Average Test Times

9.3 Preliminary Test 1 – Random or Sequential FEATURE_ID Generation

Random or sequential FEATURE_ID generation is an option when two FEATURE_IDs are required for query – i.e. for the *9-Intersection Pairs* test.

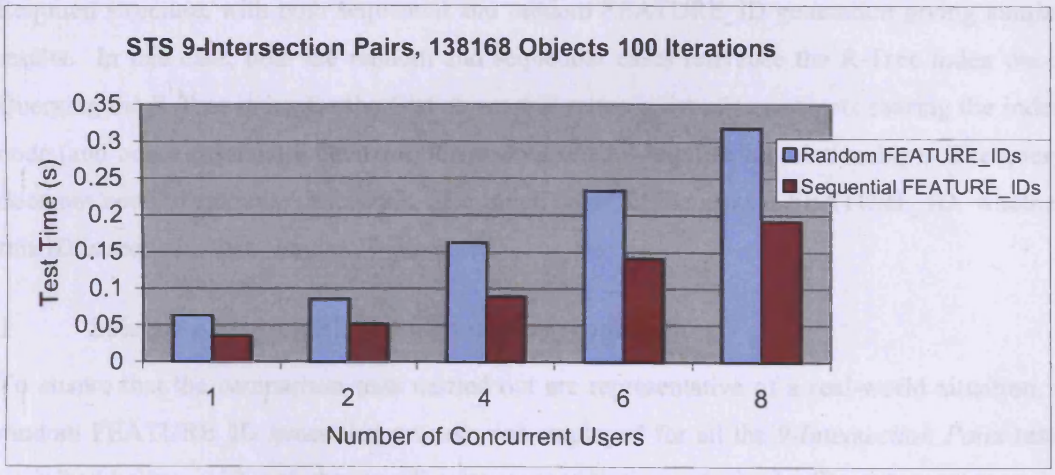


Figure 61 - STS, Random and Sequential FEATURE_IDs, 135168 objects

Figure 61 shows the results obtained when executing sequential and random FEATURE_ID generation queries for STS (similar results were obtained for 3DFDS – see Appendix 9 for data for these tests).

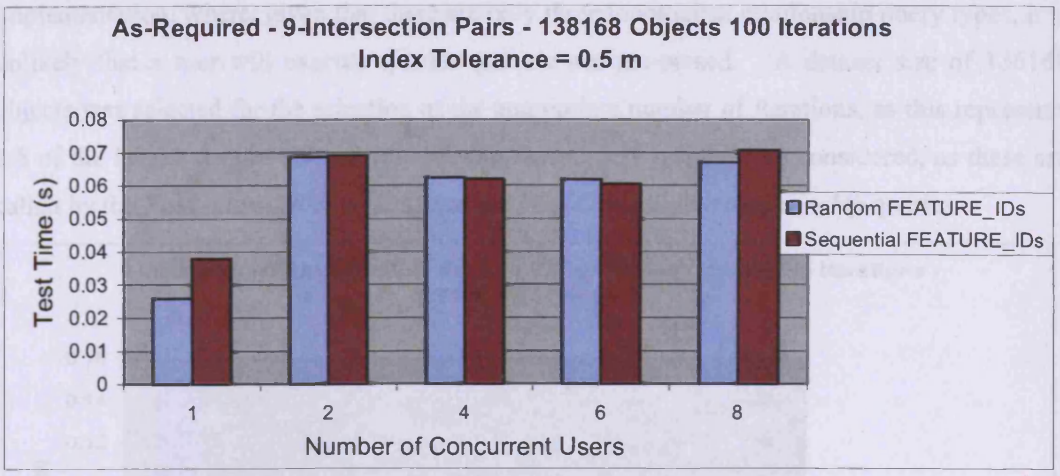


Figure 62 - Random and Sequential IDs, Proxy for As-Required, *9-Intersection Pairs*

Figure 62 gives the results obtained when executing sequential and random FEATURE_ID generation queries for the As-Required structure. Results shown here are for the 138168 object dataset.

As expected, the queries where two random FEATURE_ID values are used perform more slowly than those for sequential FEATURE_IDs for both 3DFDS and STS. For sequential FEATURE_IDs, data resides on the same disk block, requiring fewer disk read or memory operations for access. However, a similar result was not obtained for the tests against the As-Required structure, with both sequential and random FEATURE_ID generation giving similar results. In this case, both the random and sequential cases reference the R-Tree index once. Querying the R-Tree index for the first object will return a list of any objects sharing the index node (and hence potentially having a non-disconnected 9-Intersection relationship). The query does not need to retrieve a second R-Tree index node for the second FEATURE_ID, whether this ID is sequentially or randomly generated.

9.3.1 Selecting FEATURE_ID Generation Approach

To ensure that the comparison tests carried out are representative of a real-world situation, a random FEATURE_ID generation process was employed for all the *9-Intersection Pairs* tests described below. Although the overall query execution times obtained will not be as fast as for sequential data, this ensures that multiple disk reads are required for queries.

9.4 Preliminary Test 2 – Number of Test Iterations

Iterating over a query ensures that the SQL statement is parsed and pinned in memory, eliminating parse time from the query execution time (see Appendix 7 for a description of the process of executing queries in Oracle). Running the tests in this manner emulates a real-world

implementation, where, given that there are only three topological relationship query types, it is unlikely that a user will execute queries that are not pre-parsed. A dataset size of 135168 objects was selected for the selection of the appropriate number of iterations, as this represents 1/8 of the largest dataset size. *9-Intersection Pairs* query results were considered, as these are called by the *Find Intersecting Objects* and the *Find Objects with Relationship* queries.

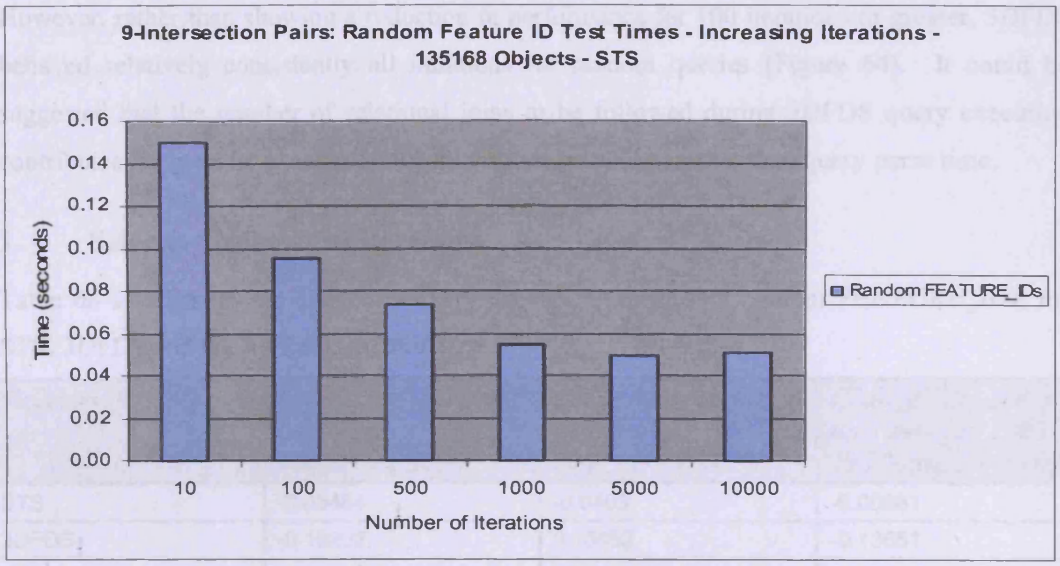


Figure 63 - STS *9-Intersection Pairs*, varying Iterations, 135168 Objects

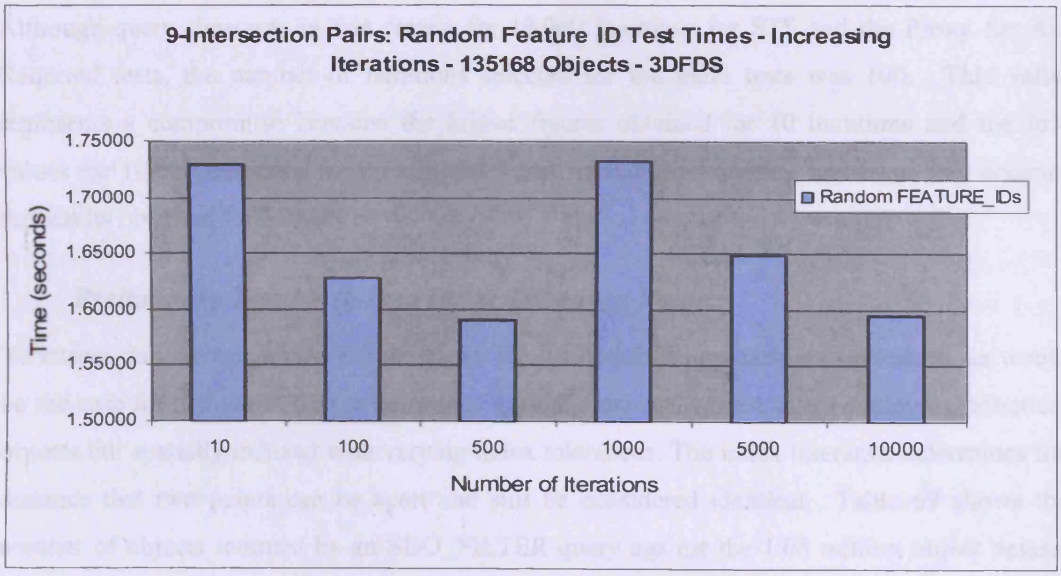


Figure 64 - 3DFDS *9-Intersection Pairs*, varying Iterations, 135168 Objects

As expected (Figure 63) execution time is higher for lower iterations for STS. A similar trend was also observed for the Proxy for the As-Required queries (data is given in Appendix 9). This matches the prediction given in Chapter 8, which hypothesised that, due to the initial query parsing process whereby Oracle determines an execution plan for the SQL and stores it in

memory, a higher value will be obtained for the execution time for single iteration tests than others. Once the execution plan is pinned in memory, average measured performance for individual queries over higher numbers of iterations will converge. Performance time per query measured at, for example, 100, 1,000 or 10,000 iterations is similar.

However, rather than showing a reduction in performance for 100 iterations or greater, 3DFDS behaved relatively consistently all iterations for random queries (Figure 64). It could be suggested that the number of relational joins to be followed during 3DFDS query execution contributes the greatest element of overall performance time, rather than query parse time.

9.4.1 Selecting Number of Iterations

Table 68 summarises the change in query execution time, using random FEATURE_IDs, for STS, 3DFDS and the As-Required structure.

Structure	*Change in execution time between 10 and 100 iterations (s)*	*Change in execution time between 100 and 1000 iterations (s)*	*Change in execution time between 1000 and 10,000 iterations (s)*
STS	-0.05484	-0.0403	-0.00381
3DFDS	-0.10202	0.10453	-0.13651
As-Required	-0.21219	-0.02693	-0.00274

Table 68 – Variation in Query Execution Time, 10 and 100 Iterations, *9-Intersection Pairs*

Although query time was in fact fastest for 10,000 iterations for STS and the Proxy for As-Required tests, the number of iterations selected for the main tests was 100. This value represents a compromise between the higher figures obtained for 10 iterations and the low values for 10,000 iterations for both the STS and As-Required queries, and takes into account the results obtained for 3DFDS.

9.5 Preliminary Test 3 – R-Tree Index Tolerance Value

To ensure that the test results for the Proxy for As-Required approach are optimised, as would be the case for real world data, a number of queries were run against tables containing identical objects but spatially indexed with varying index tolerances. The index tolerance determines the distance that two points can be apart and still be considered identical. Table 69 shows the number of objects returned by an SDO_FILTER query against the 1.08 million object dataset for each index tolerance for the above tests.

Index Tolerance (m)	*Objects Returned*	*Random Object Query Time (s)*	*Index Size (MB)*
0.05	1	*0.0823*	164.5657
0.5	1	0.0760	164.5657
1	1	0.0778	164.5657
5	8	0.1078	164.5657
100	10751	0.1175	120.9134
500	51711	0.3130	120.7551

Table 69 – Approximate no. of Objects Returned, As-Required R-Tree Query

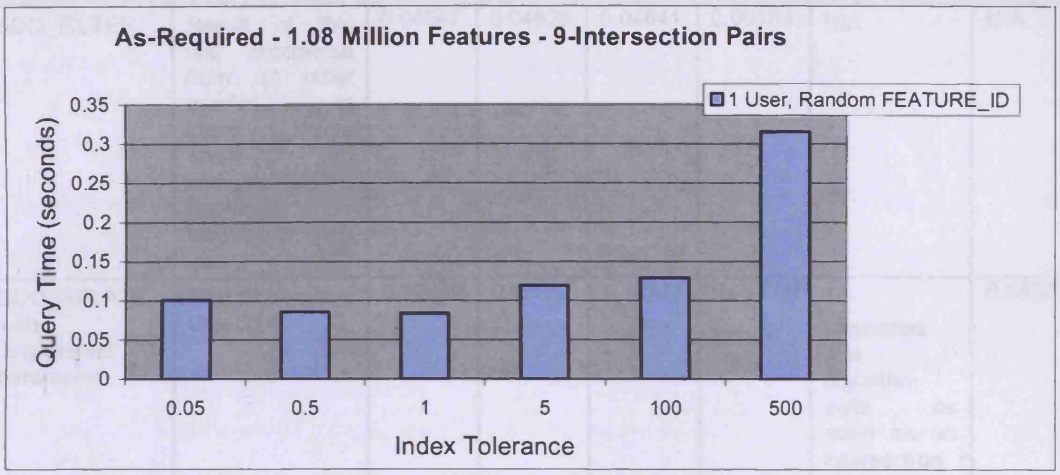


Figure 65 - Query Execution Times – 1 User - Varying R-Tree Index Tolerances

Figure 65 shows per query execution times for varying index tolerances for the 1.08 million object dataset with 1 user. Data corresponding to the above graph can be found in Appendix 9. As can be seen, query execution time is lowest where queries return single objects, even though the overall index size for these tolerances is larger than that for higher values. Test times for the 500m tolerance index are 2.6 times that for the 100m tolerance, even though five times as many objects are returned. Test times for the 100m tolerance are, however, only 1.09 times that of the 5m tolerance, even though the number of objects returned is approximately 1000 times as great. The results described here did not match those predicted in Chapter 8. Performance is not dependant on index size as predicted but instead may be dependant on number of returned objects.

9.5.1 Selecting a Tolerance Value

Following of review of the results obtained, the 0.5m index tolerance was selected for further comparison as fastest query performance was obtained for this value.

9.6 *Preliminary Test 4 – Improving the Proxy for As-Required Queries*

This test determines the overhead for relationship identification against an As-Required data structure. Table 70 presents a description of each test and the associated results. One hundred iterations of each query were carried out against the MasterMap data, and three sets of tests were run. Randomly generated FEATURE_IDs were used for all tests, and the execution times recorded include the time to generate these random IDs.

Queries Execute	*Result used as Proxy For*	*Min (s)*	*Max (s)*	*Avg (s)*	*StdDev*	*3D Added Value*[15]	*Result (added to 3D Proxy)*
SDO_FILTER	Result of this test subtracted from all other queries to identify time spent on computational geometry element of each query.	0.04547	0.04829	0.04641	0.00163	N/A	N/A
SDO_RELATE with 'anyinteract' parameter	*Find Intersecting Objects*	*0.09469*	*0.12672*	*0.10578*	*0.01814*	1 (Assumes that algorithm exits as soon as an intersection is identified)	*0.05937*
SDO_RELATE with 'touch' parameter	*Find Objects With Relationship*	*0.26110*	*1.13269*	*0.55975*	*0.49633*	1 (Multiply by 6 to account for 3D data, divide by 6 to account for each object touching 6 others in the 2D dataset)	*0.51334*
SDO_RELATE with the 'determine' parameter	*9-Intersection Pairs*	*0.02172*	*0.02266*	*0.02208*	*0.00051*	6 (Multiply by 6 to account for 3D data)	*0.13248*

Table 70 – Results - 2D Spatial Queries in Oracle

As expected the SDO_RELATE (ANYINTERACT and TOUCH) queries took longer to execute than the SDO_FILTER queries. TOUCH is also slower than ANYINTERACT, as the latter returns a result immediately some form of intersection is identified whereas the former is required to evaluate the intersection and determine whether the relationship is actually TOUCH.

[15] As described in Section 8.4.4, this improves the Proxy by accounting for the additional primitives due to 3D data.

The results obtained for the DETERMINE query were markedly more rapid than those obtained for the other SDO_RELATE queries. This could due to a combination of two factors – firstly, the DETERMINE query involves two known FEATURE_IDs and thus may not reference the R-Tree index. Secondly, the use of random FEATURE_ID generation for the queries meant that it is possible that many of the object pairs selected were DISJOINT[16].

It is important to consider the use of worst-case algorithm complexity to determine the multiplication factor when examining these results (see Section 8.4.4). Results obtainable in practice may vary depending on the actual algorithms used to implement the 2D relationship queries by Oracle and the selected algorithm for 3D topological engine implementation.

9.7 Main Tests

The main tests executed to support the structure comparison are summarised in Table 71 below. Each test was run 100 times, and repeated for three iterations to account for the shared database server and network.

Test Name	*Test Type*	*Data Structure*	*Data Volume*	*User Numbers*
Scalability	Constant	Constant	Constant	Variable
Algorithm Time Complexity	Constant	Constant	Variable	Constant
Impact of different workloads	Variable	Constant	Constant	Constant

Table 71 – Comparative Performance Tests

Summary data for the graphs shown here is given in Appendix 10, and raw data can be found on the attached CD. As with the preliminary tests, all results relating to the Proxy for As-Required tests are based on the use of worst-case algorithm complexity to determine the multiplication factor (see Section 8.4.4). Results obtainable in practice may vary depending on the complexity ratio between the proprietary algorithms used to implement the 2D relationship queries by Oracle and the selected algorithm for 3D topological engine implementation.

9.8 Main Test 1 – Scalability

Scalability tests examine the impact of increasing numbers of users running concurrent queries. Results obtained for each of the three data structures, *9-Intersection Pairs* tests, are described below, followed by a comparison with the expected results detailed in Chapter 8. Results for

[16] Investigation into why the 'determine' query performs more rapidly than the SDO_FILTER operation is ongoing with Oracle (Service Request Number 6073112.994). To avoid negative values, the SDO_FILTER value was not subtracted from the 'determine' result in this case.

the *Find Intersecting Objects* and the *Find Objects with Relationship* tests against the individual structures, which showed similar trends to the *9-Intersection Pairs* test results, are also given in Appendix 10. Comparative results for all three structures are presented here.

9.8.1 Scalability – Individual Structures

Figure 66, Figure 67 and Figure 68 graph the performance of the *9-Intersection Pairs* queries for increasing numbers of concurrent users.

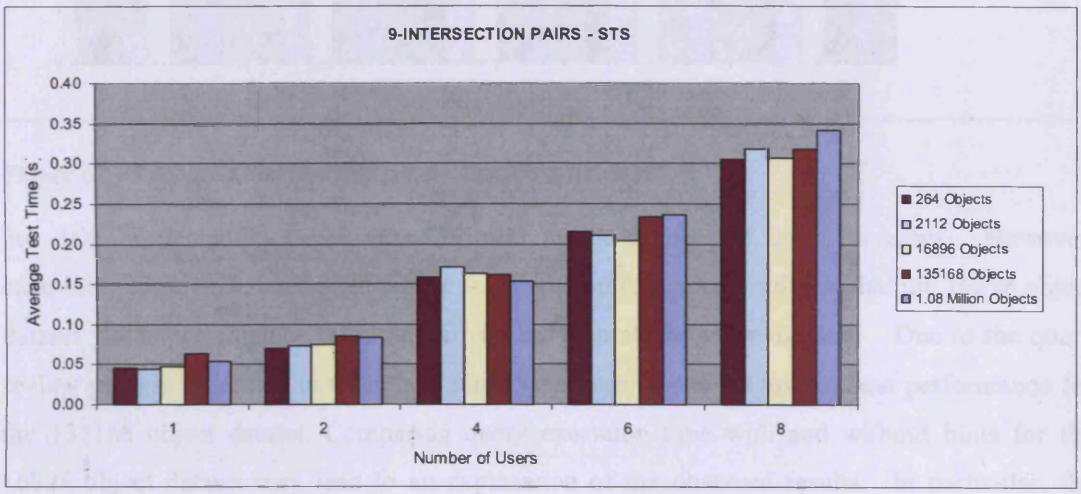


Figure 66 - Scalability Tests for STS, *9-Intersection Pairs*

As predicted, the execution time of the *9-Intersection Pairs* queries appears to increase in a linear manner as the number of users increases. However, the expected inflexion between 4 and 6 concurrent users, which was predicted to occur due to the 4-processor server, cannot be clearly identified. This implies that, for the tests conducted, the processor was not a source of contention. Given the 4-CPU server, it may also be expected that constant performance would be observed for 1-4 users. However, other forms of contention may be an issue – including disk contention or memory access. This is particularly the case in STS as the queries are not join intensive (which would place additional load on the CPU due to the algorithms required to determine the join relationships).

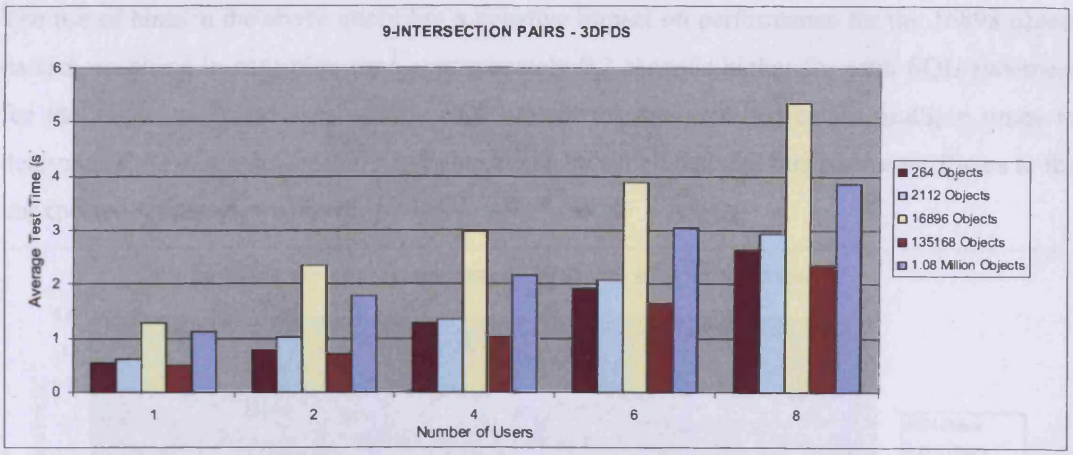


Figure 67 - *9-Intersection Pairs* Scalability Test Results, 3DFDS

For 3DFDS, scalability again appears linear as the number of users increases. However, examining Figure 67 above identifies an issue with performance results, in that the 16898 object dataset yields performance times that are higher than all the other datasets. Due to the query review process described in Chapter 7 hints have been optimised toward best performance for the 135168 object dataset. Comparing query execution time with and without hints for the 16898 object dataset may lead to an explanation of the observed results. In particular, the following query is of interest:

```
SELECT COUNT(*) FROM (SELECT /*+ORDERED */ A.NODE_ID
FROM TABLE(GEN_SHARE_NODE.GET_ALL_NODES(23)) A
RIGHT OUTER JOIN
GEN_TOPO_PART_TABLE B
ON A.PARENT_TOPO_ID = B.PARENT_TOPO_ID
WHERE B.GEOMETRY_ID = 23 AND A.IS_BOUNDARY = 0
MINUS
SELECT /*+ORDERED */ B.NODE_ID
FROM TABLE(GEN_SHARE_NODE.GET_ALL_NODES(24)) B
RIGHT OUTER JOIN
GEN_TOPO_PART_TABLE C
ON B.PARENT_TOPO_ID = C.PARENT_TOPO_ID
WHERE C.GEOMETRY_ID = 24)
```

Table 72 shows the results of the comparison for all datasets for 3DFDS.

Number of Objects	*Query Execution Time (s) with Hints*	*Query Execution Time (s) without Hints*
264	0.56	0.49
2112	0.56	0.53
16898	0.83	0.56
135168	0.53	0.67

Table 72 – 3DFDS – *9-Intersection Pairs* - SQL Execution Time

The use of hints in the above query has a negative impact on performance for the 16898 object dataset, resulting in execution time approximately 0.2 seconds higher for each SQL statement for the single user case. As similar SQL statements are executed called multiple times to determine the *9-Intersection Pairs* R-Value, it can be concluded that this issue contributes to the unexpected results shown above.

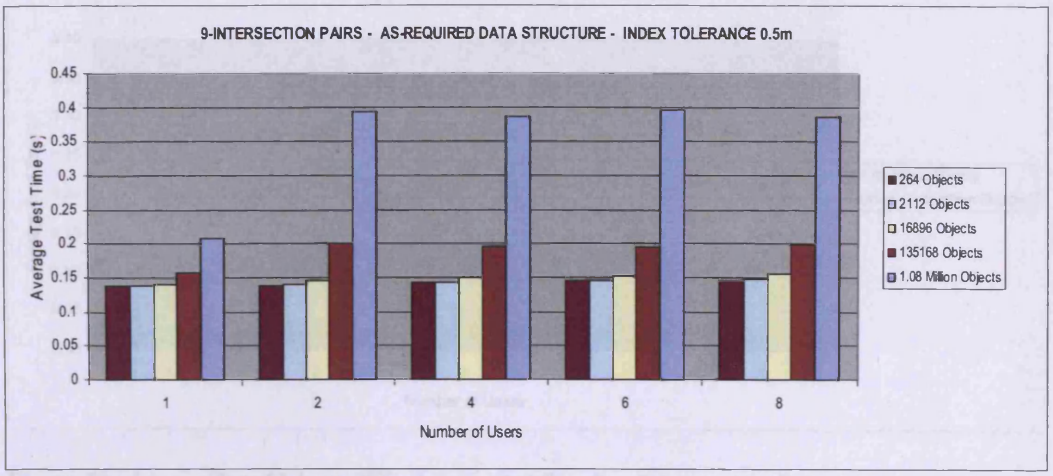


Figure 68 - Scalability Test Results - Proxy for As-Required Queries

The As-Required query results show no performance drop as the number of users increases. This may be due to the use of a 3D SDO_FILTER query as a Proxy for this test. Running this test will result in one query against the R-Tree index (find the R-Tree index entry for FEATURE_ID 1, and check this entry for FEATURE_ID 2). Additionally, the index tolerance selected (0.5m) returns a maximum of 1 intersecting object per index Node, no matter the size of the dataset. Additionally, the 3D added value (Table 70) was only determined for the single user case.

9.8.2 Trend Analysis

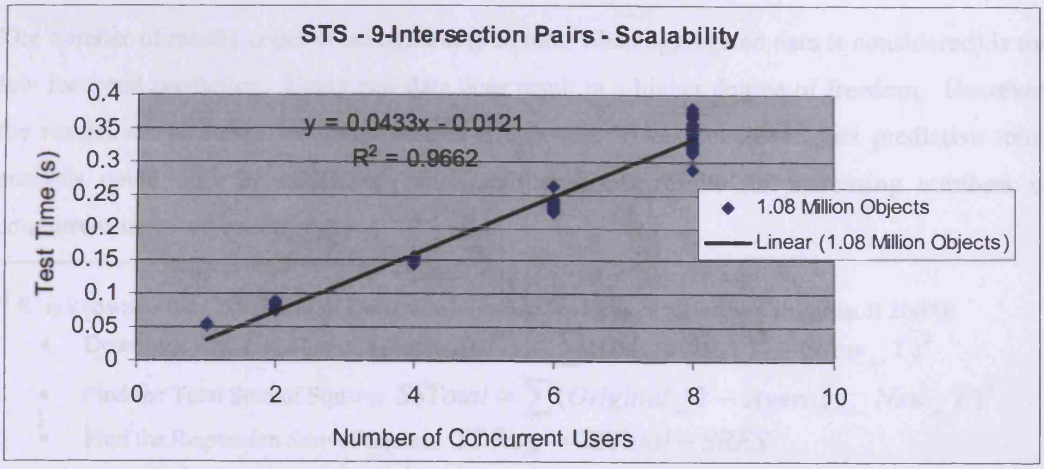


Figure 69 - Linear Trend Lines, STS, Raw Results, *9-Intersection Pairs*

Microsoft Excel was used to generate trend lines (and hence predict scalability) for the results obtained. A sample trend line for raw data is shown in Figure 69, and for aggregated data in Figure 70.

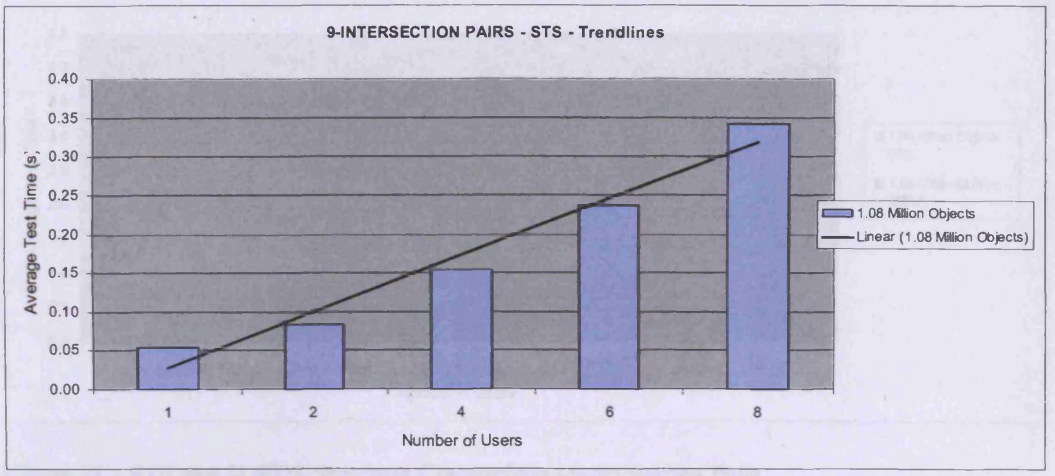


Figure 70 - Linear Trend Lines, STS, Aggregated Results, *9-Intersection Pairs*

Table 73 shows the trend lines and the corresponding Coefficient of Determination (R^2) values[17], where 1 indicates a perfect fit. Both linear and polynomial (Order 2) trend lines give a very good match for this data. The high values of R^2 obtained are however due to the low number of degrees of freedom (number of data points minus number of fitted parameters).

Dataset Size	*Data Source*	*Trend Line*	*R^2*	*Degrees of Freedom*
1.08 Million	Raw	y =0.0433x - 0.0121	0.9662	62
1.08 Million	Raw	y =0.0022 x^2 + 0.0211x + 0.032	0.9765	61
1.08 Million	Aggregated	y = 0.0407x + 0.0024	0.9947	3
1.08 Million	Aggregated	y = 0.0021x^2 + 0.0218x + 0.0306	0.9999	2

Table 73 - Trend Analysis, Microsoft Excel

The number of results under consideration (5 in total when aggregated data is considered) is too few for trend prediction. Using raw data does result in a higher degree of freedom. However, the results are clustered vertically around five points. It was concluded that predictive trend analysis could only be undertaken once additional test results for increasing numbers of concurrent users are executed.

---

[17] R^2 is known as the Coefficient of Determination and is calculated as follows (Microsoft 2007):

- Determine Residual Sum of Squares $SRES = \sum (Original\_Y)^2 - (New\_Y)^2$
- Find the Total Sum of Squares $SSTotal = \sum (Original\_Y - Average\_New\_Y)^2$
- Find the Regression Sum of Squares $SS\,\mathrm{Re}\,g = SSTotal - SRES$
- Then: $R^2 = SS\,\mathrm{Re}\,g / SSTotal$

9.8.3 Scalability – Comparing Structures – STS and 3DFDS

Figure 71, Figure 72 and Figure 72 below show comparisons for the three queries, for 1.08 Million objects for STS and 3DFDS.

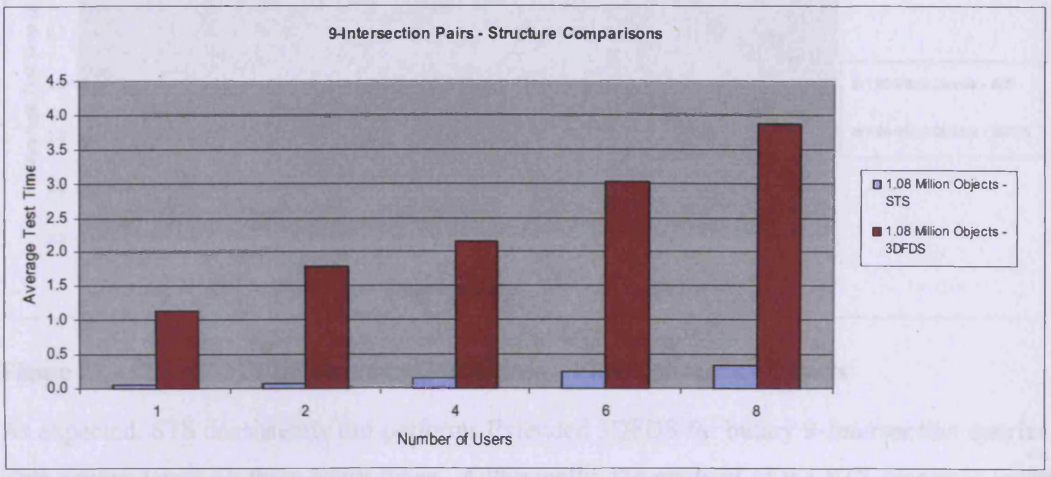


Figure 71 – STS and 3DFDS Structure Comparison –*9-Intersection Pairs*

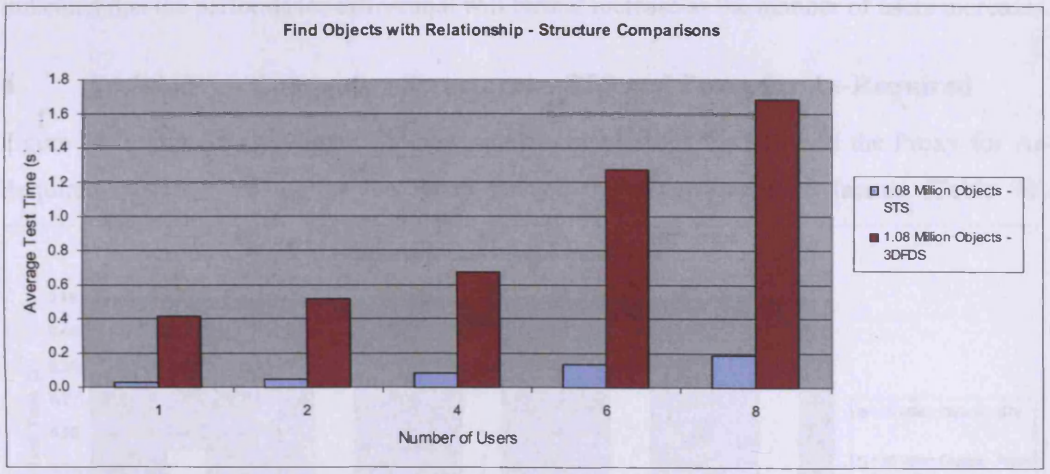


Figure 72 – STS and 3DFDS Comparison – Find Objects with Relationship

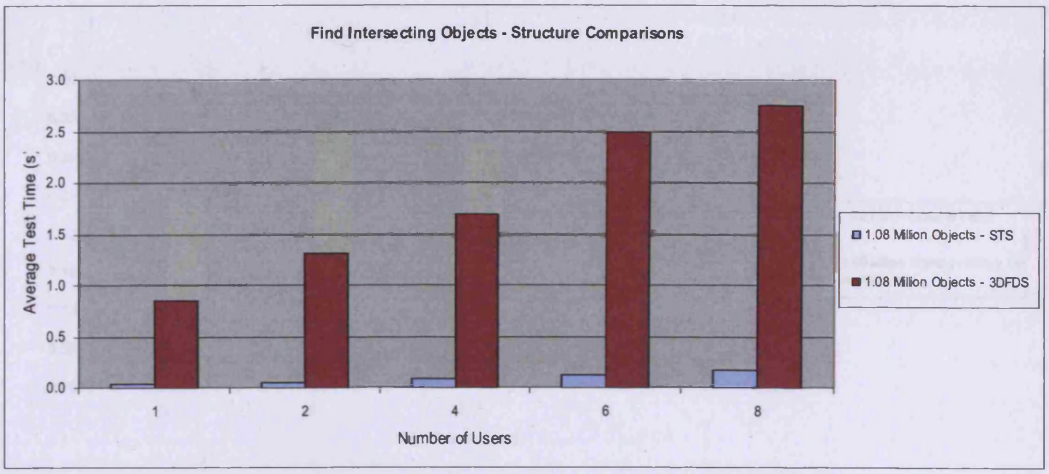


Figure 73 – STS and 3DFDS Structure Comparison – Find Intersecting Objects

As expected, STS consistently out-performs Extended 3DFDS for binary 9-*Intersection* queries. This occurs across all three query types. Additionally, the gradient of the STS graphs is lower than that for 3DFDS – i.e. the drop in performance with increasing numbers of users is less indicated that the performance differential will further increase as the number of users increases.

9.8.4 Scalability – Comparing Structures – STS and Proxy for As-Required

Figure 74, Figure 76 and Figure 75 show similar comparisons for STS and the Proxy for As-Required queries. Values for the Proxy include the 3D maintenance factors (Table 70).

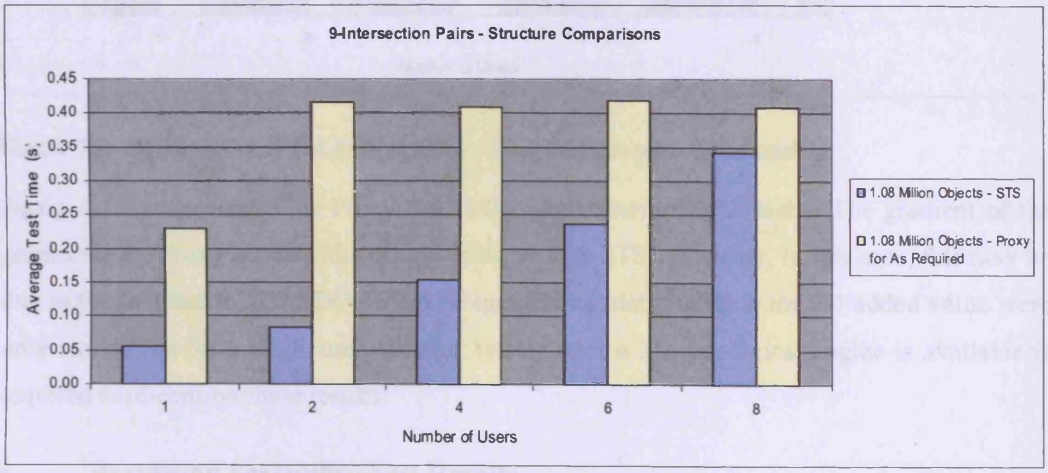


Figure 74 - STS and Proxy for As Required - Structure Comparison – 9-Intersection Pairs

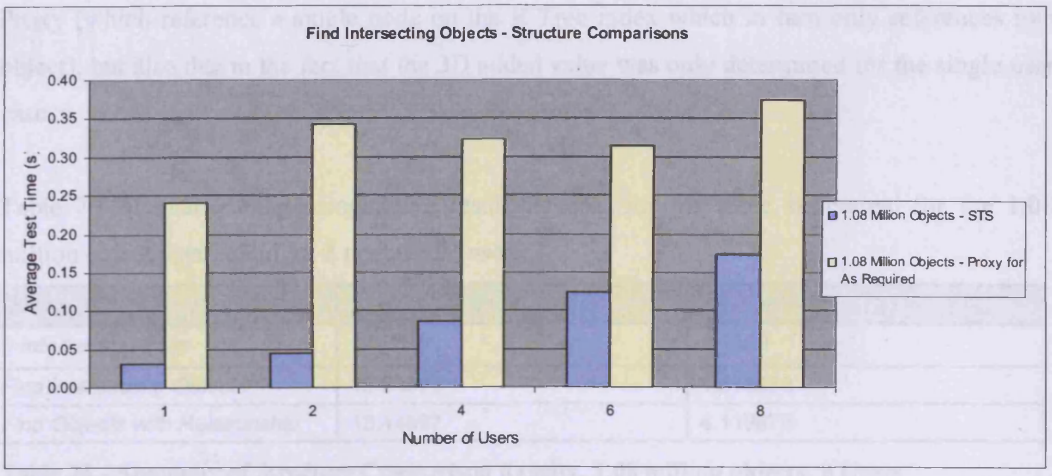


Figure 75 – STS and Proxy for As-Required - Find Intersecting Objects

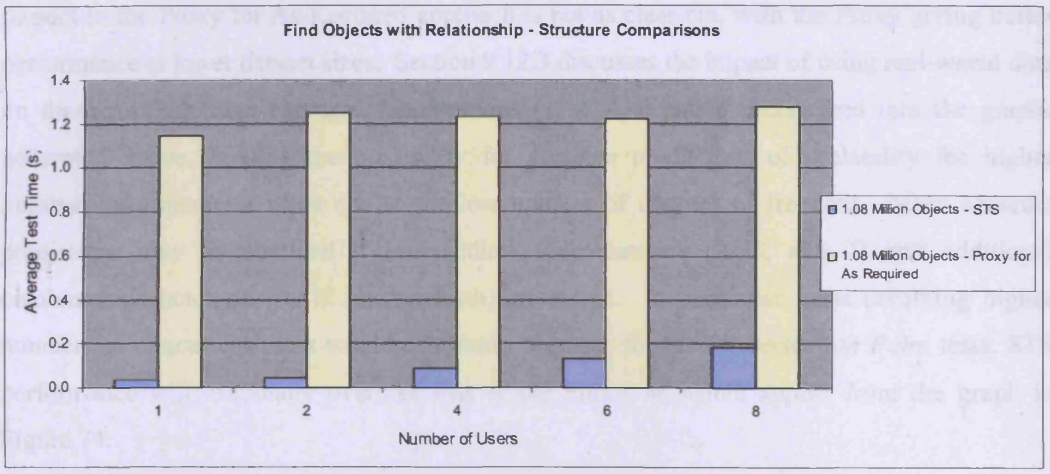


Figure 76 – STS and Proxy for As-Required – Find Objects with Relationship

Again, STS out-performs the Proxy for As-Required queries for all tests. The gradient of the graphs for the Proxy appears identical to or better than STS. However, in this case, this may be due to the fact that the 2D SDO_RELATE query tests contributing to the 3D added value were only carried out for a single user. Further testing once a 3D topological engine is available is required to re-confirm these results.

9.8.5 Reviewing Scalability Test Results

Predicting the scalability of an IT system is a complex task and is generally undertaken by observation and benchmarking rather than theoretical assessment. The results presented here go some way to indicating a trend for all three structures, with apparent linear scalability in all cases if anomalous results are ignored. As expected, query performance worsened as the number of concurrent users increased, with the exception of the results given for the Proxy for As-Required tests. This was due in part to the nature of the 3D queries forming part of the

Proxy (which reference a single node on the R-Tree index which in turn only references two object), but also due to the fact that the 3D added value was only determined for the single user case.

Table 74 summarises the comparative results (ratios) for the three structures, for the 1.08 million object dataset and for 8 concurrent users.

Structure	*3DFDS/STS*	*As-Required/STS*
9-Intersection Pairs	11.3316	1.1329
Find Intersecting Objects	15.94876	1.818363
Find Objects with Relationship	15.44897	4.119875

Table 74 – Summary of Structure Comparison Results, 1.08 million objects, 8 Users

As expected, STS performed better than 3DFDS in all cases. However, the situation with respect to the Proxy for As-Required approach is not as clear-cut, with the Proxy giving better performance at lower dataset sizes. Section 9.12.3 discusses the impact of using real-world data on these results. Five aggregate observations (1, 2, 4, 6 and 8 users) feed into the graphs generated above, limiting the possibility for accurate predictions of scalability for higher numbers of concurrent users due to the low number of degrees of freedom. More accurate predictions may be obtained if intermediate user numbers (3, 5, and 7) and additional combinations (such as 10, 12 and so forth) are tested. In particular, tests involving higher numbers of concurrent users would determine whether, for the *9-Intersection Pairs* tests, STS performance will eventually overtake that of the Proxy, as would appear from the graph in Figure 74.

9.9 Main Test 2 - Algorithm Time Complexity

Algorithm time complexity tests measure query performance the performance of with increasing data volume. Results are first presented on a structure-by-structure basis for the *9-Intersection Pairs* tests, with data given in Appendix 10. Comparative results are then given for STS and 3DFDS and for STS and the Proxy. Note that due to the range in dataset sizes (264 to 1.08 million objects) a logarithmic scale has been applied to the x-axis of the graphs.

9.9.1 Algorithm Time Complexity – Individual Results

Figure 77, Figure 79 and Figure 81 show algorithm complexity test results for the STS, 3DFDS and Proxy for As-Required structures respectively, for the *9-Intersection Pairs* queries.

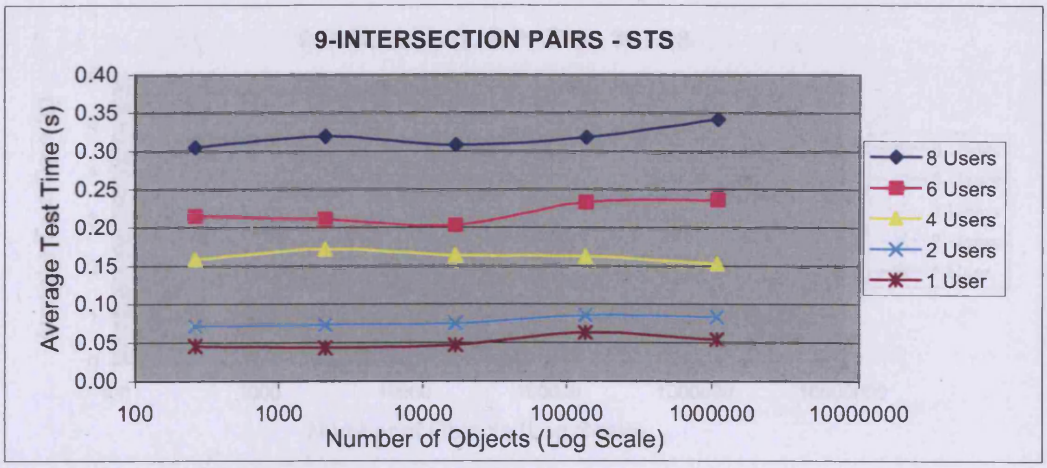


Figure 77 - *9-Intersection Pairs*, STS, Algorithm Time Complexity

Predicted algorithm complexity for this test for STS was given as O(sqrt(n)). Figure 78 shows a square root function where n is the number of objects, along with the results for the 8 user tests for the *9-Intersection Pairs* query (the square root values have been scaled down by a factor 1000). The algorithm complexity results for STS are thus better than expected, showing near constant time.

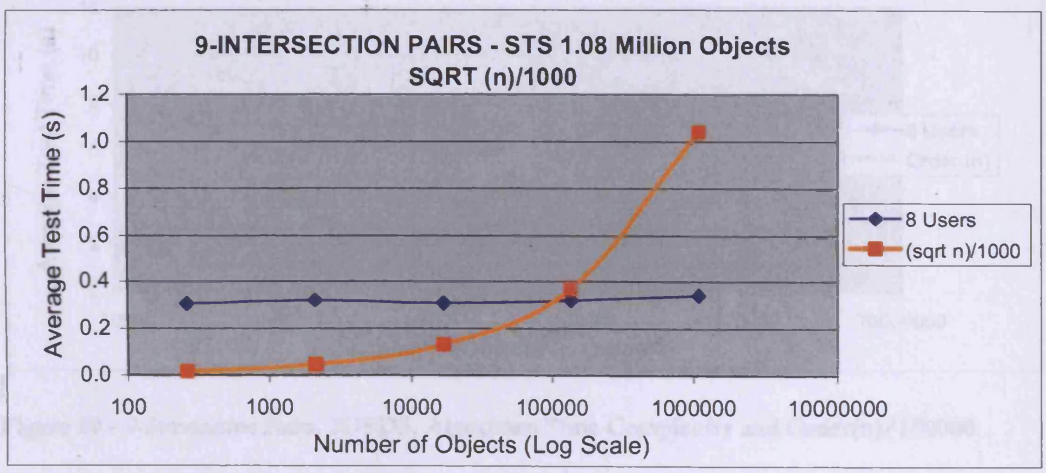


Figure 78 - *9-Intersection Pairs*, STS, Algorithm Time Complexity and Sqrt(n)/1000

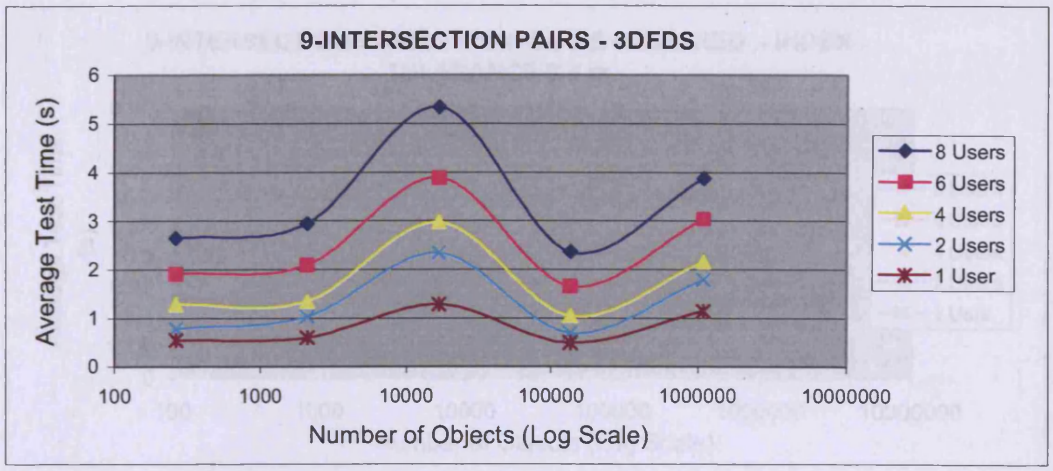


Figure 79 - 3DFDS - *9-Intersection Pairs* - Algorithm Time Complexity Results

The graph in Figure 79 shows Algorithm Time Complexity for 3DFDS. Again, deterioration in performance can be observed for the 16898 object queries. This is consistent across all user combinations, and corresponds to the issues observed during the scalability tests.

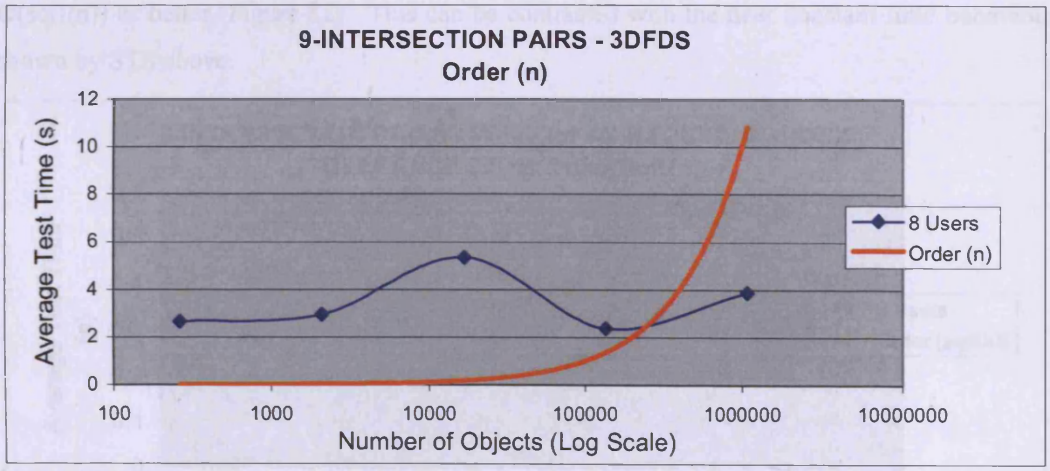


Figure 80 - *9-Intersection Pairs*, 3DFDS, Algorithm Time Complexity and Order(n)/100000

Given this, it is not possible to compare algorithm complexity to that expected (i.e. O(n) although again it would appear that results obtained are better than predicted. Figure 80 shows a graph showing the 8-User data for 3DFDS with an Order(n) function.

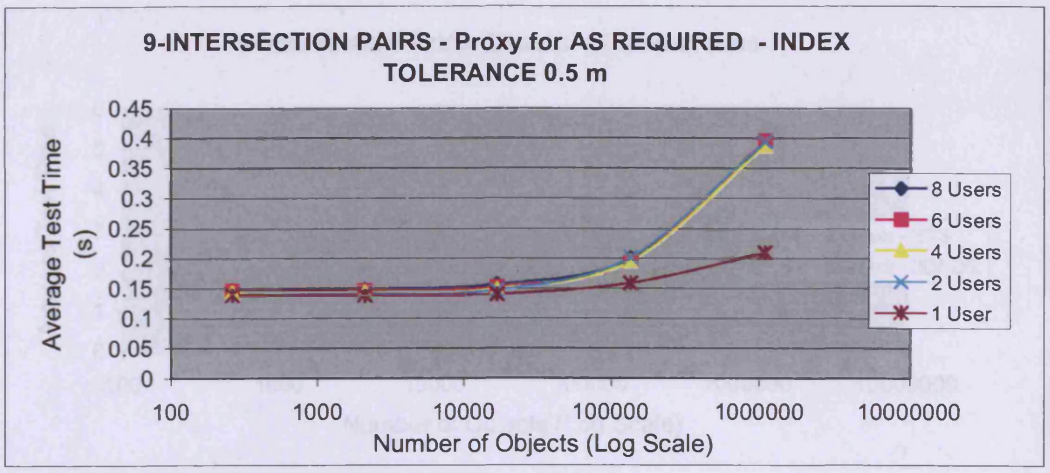


Figure 81 – As-Required - *9-Intersection Pairs* - Algorithm Time Complexity Results

As discussed in Chapter 8, due to the use of commercial software where the internal algorithms are unpublished, it was not possible to predict any trends in algorithm time complexity for the Proxy for As-Required tests. However, the above graph appears to show behaviour of O(sqrt(n)) or better (Figure 82). This can be contrasted with the near constant time behaviour shown by STS above.

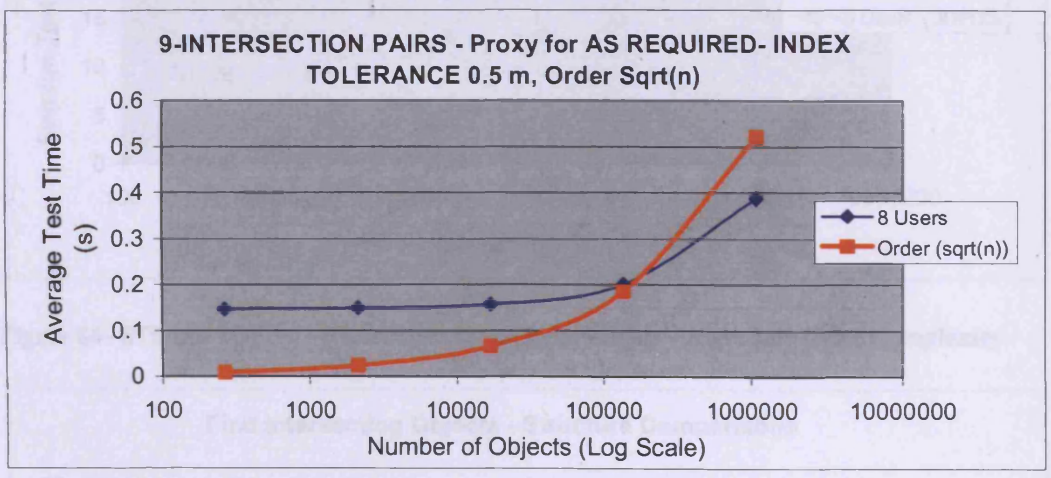


Figure 82 – As-Required - *9-Intersection Pairs* - Algorithm Time Complexity vs. O(sqrt(n))

9.9.2 Algorithm Time Complexity - Comparing Structures – STS and 3DFDS

Figure 83, Figure 84 and Figure 85 show comparisons for algorithm complexity for STS and Extended 3DFDS. All comparisons are given for the 8-user case.

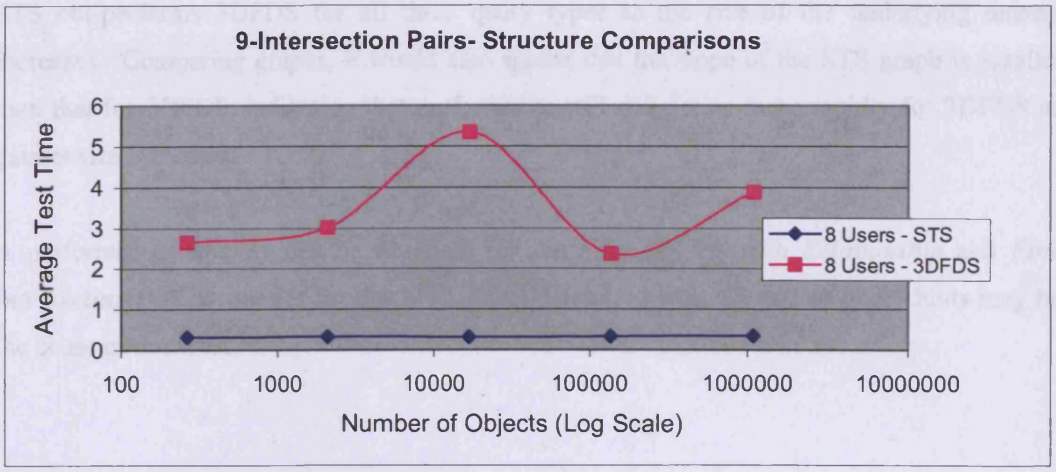


Figure 83 – STS and 3DFDS - *9-Intersection Pairs* - Algorithm Time Complexity Results

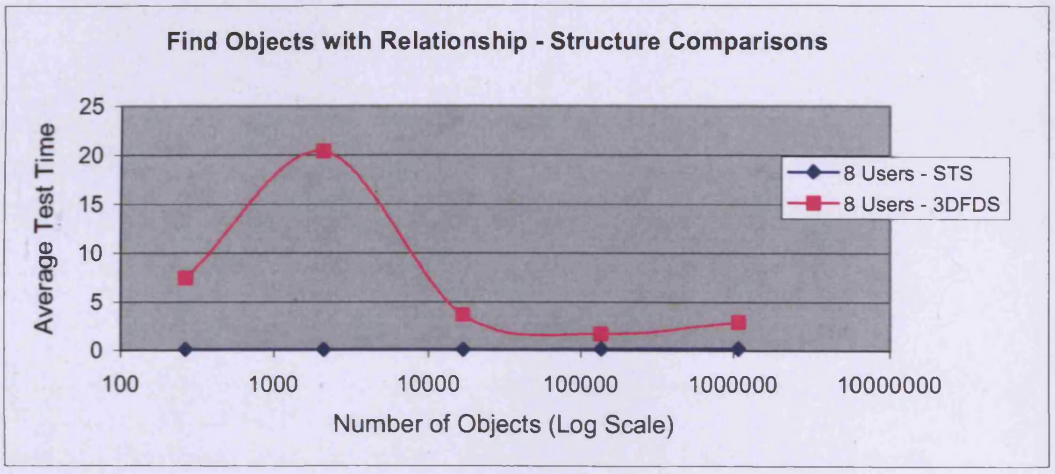


Figure 84 –STS and 3DFDS – Find Objects with Relationship - Algorithm Time Complexity

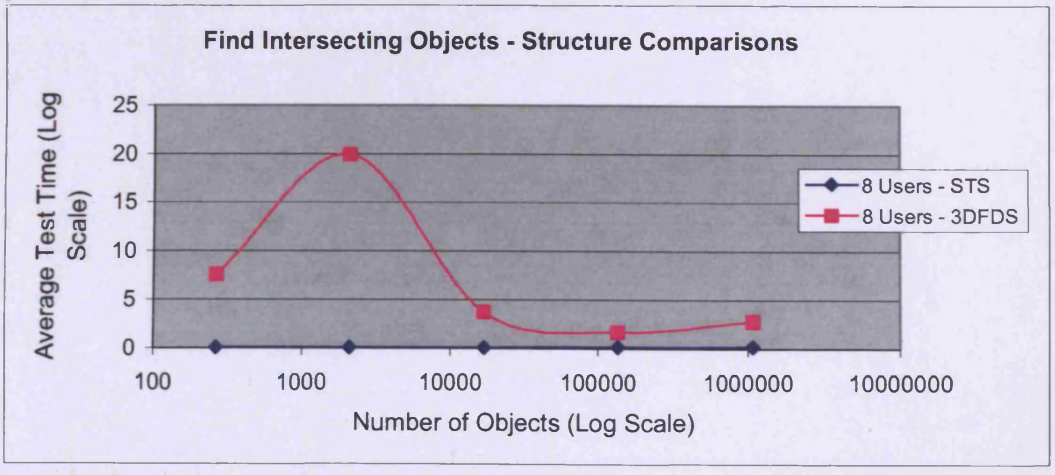


Figure 85 – STS and 3DFDS – Find Intersecting Objects - Algorithm Time Complexity

STS out-performs 3DFDS for all three query types as the size of the underlying dataset increases. Comparing graphs, it would also appear that the slope of the STS graph is smaller than that for 3DFDS, indicating that performance will deteriorate more rapidly for 3DFDS as dataset size increases.

A performance anomaly can be observed for the *Find Objects with Relationship* and *Find Intersecting Objects* queries for the 2112 object dataset. Again, the use of query hints may be the cause of this issue.

```
SELECT /*+ USE_NL(R GTPT)*/ GTPT.GEOMETRY_ID FROM GEN_TOPO_PART_TABLE GTPT INNER JOIN (
           SELECT /*+ USE_NL(GGF X)*/ PARENT_TOPO_ID FROM GEN_GEOM_FACE GGF INNER JOIN
           (SELECT  /*+ USE_NL(GNFE B)*/ FACE_ID FROM GEN_NODE_FACE_EX GNFE INNER JOIN
           (SELECT NODE_ID FROM TABLE(GEN_SHARE_NODE.GET_ALL_NODES(11))) B
          ON GNFE.NODE_ID = B.NODE_ID) X
          ON GGF.FACE_ID  = X.FACE_ID
          UNION
           SELECT /*+ USE_NL(GGF Y)*/ PARENT_TOPO_ID FROM GEN_GEOM_VOLUME GGF INNER JOIN
           (SELECT /*+ USE_NL(GFV X)*/ VOLUME_ID FROM GEN_FACE_VOLUME GFV INNER JOIN
           (SELECT /*+ USE_NL(GNFE B)*/ FACE_ID FROM GEN_NODE_FACE_EX GNFE INNER JOIN
           (SELECT NODE_ID FROM TABLE(GEN_SHARE_NODE.GET_ALL_NODES(11))) B
          ON GNFE.NODE_ID = B.NODE_ID) X
          ON GFV.FACE_ID  = X.FACE_ID) Y
          ON Y.VOLUME_ID = GGF.VOLUME_ID
--
--
          ON GTPT.PARENT_TOPO_ID = R.PARENT_TOPO_ID
          WHERE GTPT.GEOMETRY_ID != 11
          GROUP BY GTPT.GEOMETRY_ID
```

Executing this query in directly with and without hints for the lower sized datasets yields the following results (Table 75):

Dataset Size	*Query Time with Hints (s)*	*Query Time without Hints (s)*
135168	0.421	8
16898	1.4	2.4
2112	23	1.3
264	11	1.3

Table 75 – Comparing Query Execution Times for 3DFDS Find Intersecting Features

Query time is particularly high for the 2112 feature dataset, although the hints do provide performance improvements for the larger datasets.

9.9.3 Algorithm Time Complexity - Comparing Structures – STS and Proxy for As-Required

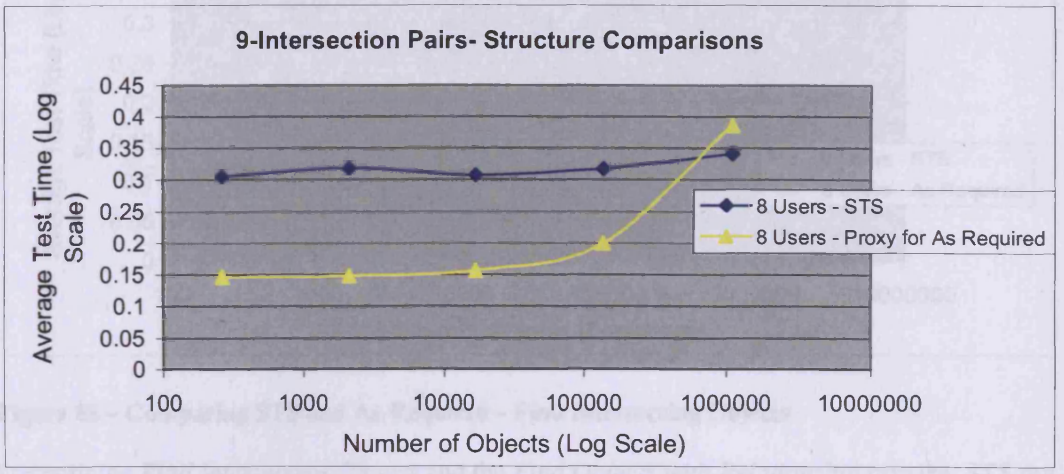


Figure 86 – Comparing STS and Proxy for As-Required, Algorithm Time Complexity

For the *9-Intersection Pairs* results, the Proxy for As-Required structure generally out-performs STS, although results appear to converge as dataset size increases and STS gives better results for the 1.08 million object dataset. The rate of performance deterioration for the As-Required Proxy is also higher than that for STS for the largest datasets, indicating that the complexity of the As-Required algorithms is higher than those for STS.

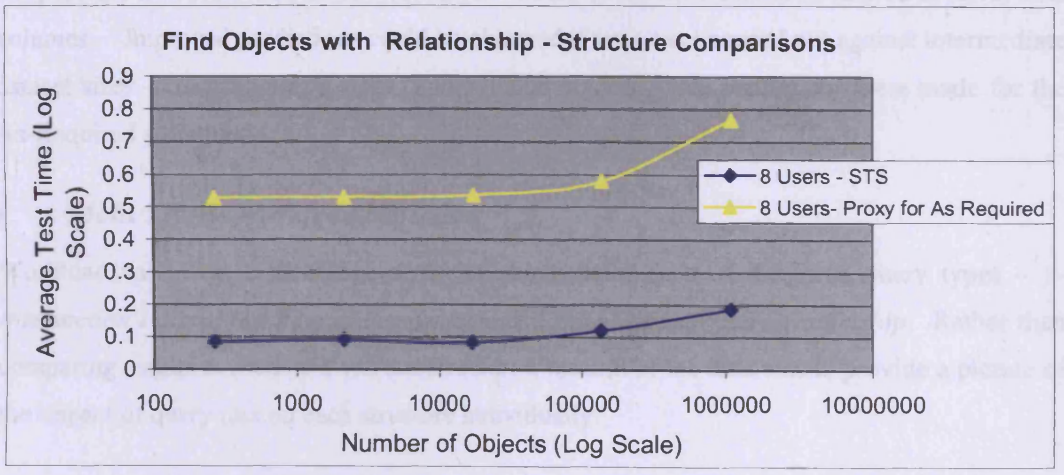


Figure 87 – Comparing STS and AS-Required - Find Objects with Relationships

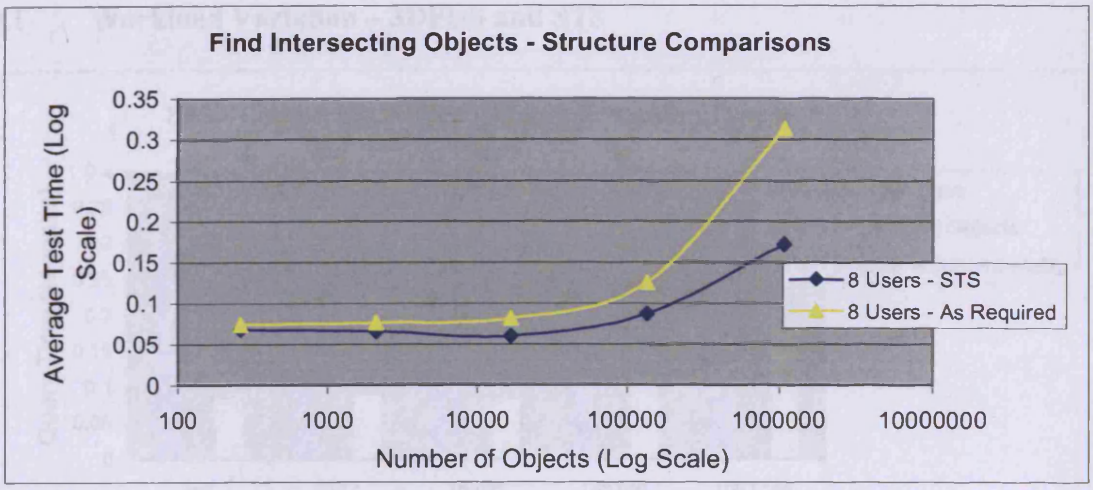


Figure 88 – Comparing STS and As-Required – Find Intersecting Objects

For both the *Find Intersecting Objects* and the *Find Objects with Relationship* queries, STS out-performs the Proxy for As-Required queries. Again, complexity for STS is lower.

9.9.4 Reviewing Algorithm Complexity Test Results

As discussed in Chapter 8, algorithm complexity depends on the implementation of the index operations within the Oracle environment, and the predictions are therefore worst case. As can be seen from the above graphs, results obtained exceeded expectations for STS, although it was not possible to reach the same conclusion for 3DFDS given the issues encountered at lower data volumes. Improved predictions could be obtained if tests were carried out against intermediate dataset sizes – for example, 270,000 and 540,000 objects. No predictions were made for the As-Required structure.

9.10 Main Test 3- Workload Variation

Workload validation tests compare results obtained for each of the three query types – *9-Intersection Pairs*, *Find Intersecting Objects* and *Find Objects with Relationship*. Rather than comparing results between the various structures, the aim of the tests was to provide a picture of the impact of query mix on each structure individually.

9.10.1 Workload Variation – 3DFDS and STS

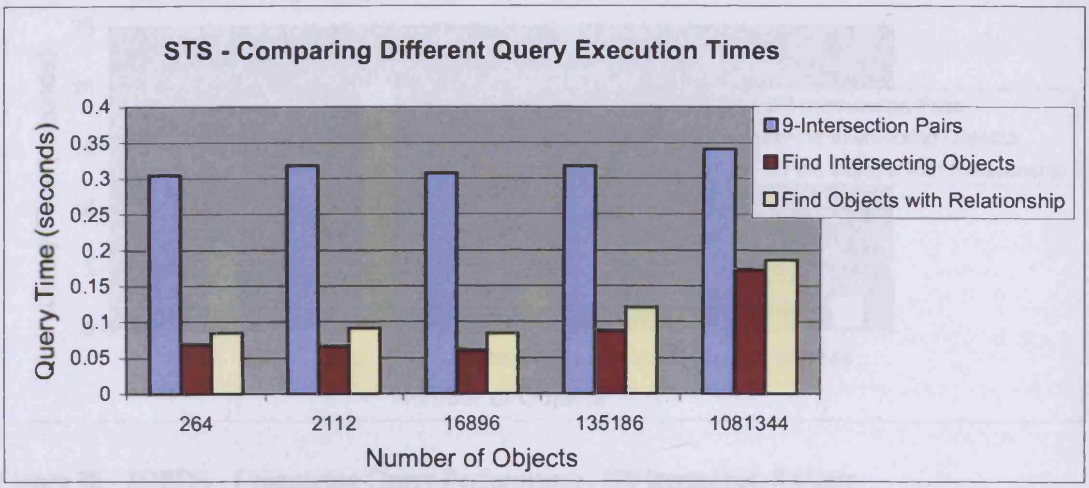


Figure 89 - STS - Comparing Query Performance, 100 Iterations, 8 Users

Figure 89 shows a graph comparing the three test results for the STS structure, for 8-User tests. As predicted, performance results obtained for the *Find Intersecting Objects* and *Find Objects with Relationship* test are similar – with the additional time for the latter being accounted for by the time required to filter out objects having the required relationship.

However, contrary to the predicted results, the *9-Intersection Pairs* test execution time exceeds both *Find Intersecting Objects* and *Find Objects with Relationship* although the latter both call the former. The order of test execution may be relevant here. As the test harness executed *9-Intersection Pairs* tests first, it is possible that these pre-loaded data into memory that was then used by the other queries. That the randomly selected objects are already pre-loaded becomes less likely as the number of objects increases, which may explain why the *Find Intersecting Objects* execution times for 264 objects are less than 1/3 of those for *9-Intersection Pairs*, where tests for 1.08 million objects take approximately 2/3 the time for the corresponding *9-Intersection Pairs* tests.

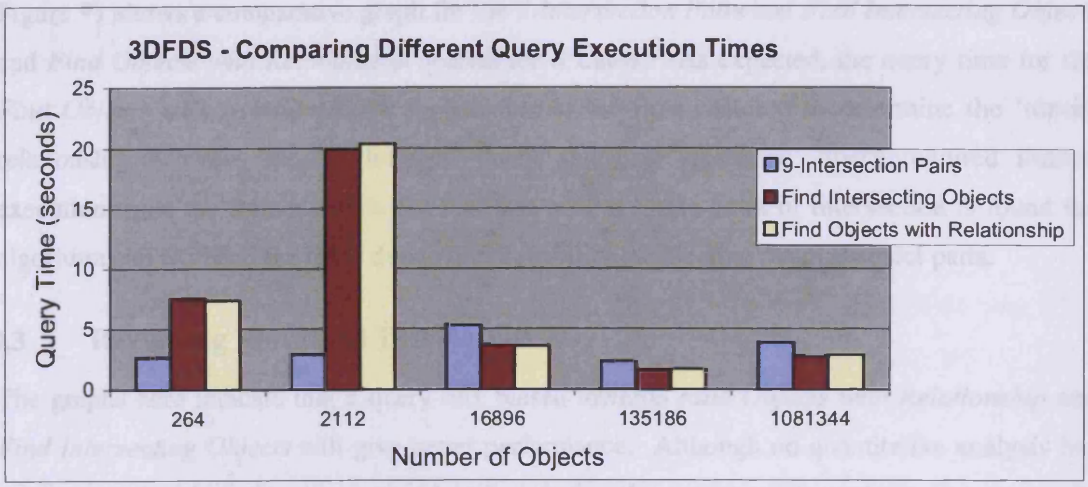


Figure 90 - 3DFDS - Comparing Query Performance, 100 Iterations, 8 Users

Figure 90 shows a graph comparing the three test results for the 3DFDS structure, for 8-User tests. Note that the high values resulting for the 2112 object dataset corresponds to the issue described above. Again, there is little difference between the results obtained for the *Find Intersecting Objects* and the *Find Objects with Relationship* queries. In this case, however, the difference between the *9-Intersection Pairs* results is not as marked as that for STS, although performance is still generally slower than that of the other two query types. It can be assumed that the length of time required to follow the join queries required to identify shared Node primitives (which form the basis of the *Find Intersecting Objects* and *Find Objects with Relationship* queries) reduces the significance of the read operation required to load data for the *9-Intersection Pairs* tests.

9.10.2 Workload Variation – As-Required Data Structure

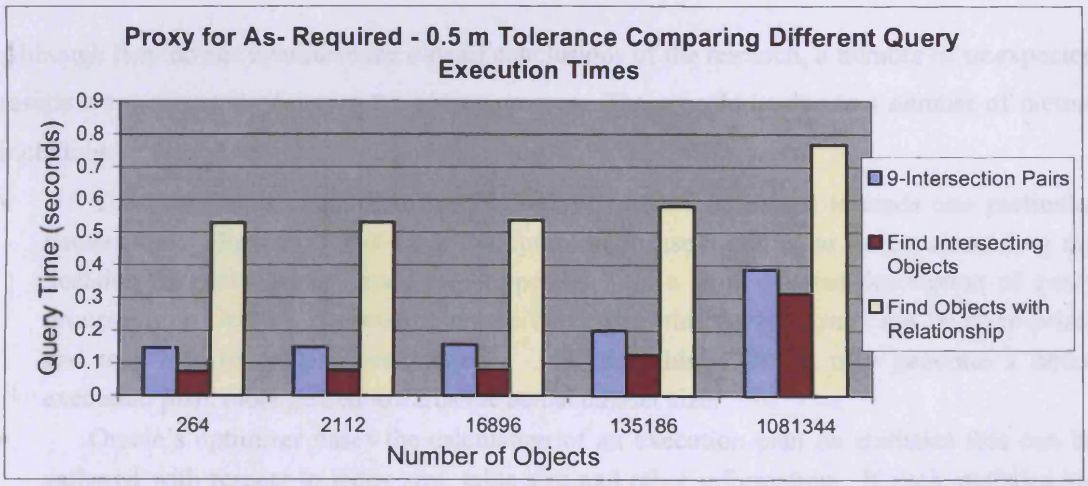


Figure 91 – Comparative Execution times, As-Required, 8 Users, 100 Iterations

Figure 91 shows a comparative graph for the *9-Intersection Pairs* and *Find Intersecting Objects* and *Find Objects with Relationship* queries for 8 Users. As expected, the query time for the *Find Objects with Relationship* is higher, due to the time required to determine the 'touch' relationship, whereas the 'anyinteract' query and the 'determine' query required limited execution time, the former due to the fact that once a single point of intersection is found the algorithm can exit, and the latter due to the probability of selecting disjoint object pairs.

9.10.3 Reviewing Workload Test Results

The graphs here indicate that a query mix biased towards *Find Objects with Relationship* and *Find Intersecting Objects* will give better performance. Although no quantitative analysis has yet been undertaken, the query mix identified during the requirements analysis process would also indicate that these relationships are more prevalent than the *9-Intersection Pairs* queries. However, given the unexpected results for the *9-Intersection Pairs* tests, further testing with real-world data and users is required to gain a better understanding of performance under mixed workload.

9.11 Comparing Results to Predictions

Reviewing the results it can be seen that they matched or bettered those predicted by theory, and no specific bottlenecks were identified for the executed tests. Scalability appears to be linear, and algorithm complexity is lower than predicted. As expected performance is rapid for small datasets and single-user tests. Additionally, the rate of deterioration of performance does increase between the 135168 object and 1.08 million object dataset, when compared with the smaller datasets. This suggests that, as planned, the highest volume dataset cannot be held in memory and that disk reads impact overall performance results for these tests.

Although they do not invalidate the overall conclusions of the research, a number of unexpected results were encountered during the testing process. These could be due to a number of factors including:

- The query hints included in the PL/SQL scripts are optimised towards one particular dataset size. Hints in Oracle force the optimizer to use a particular index, overriding the decision the optimizer has made (see Appendix 7 for a more detailed description of query processing in Oracle). Depending on the index size, this decision may not be appropriate and may lead to reduced performance. Without hints, Oracle may generate a better execution plan, more geared towards the actual dataset size.
- Oracle's optimizer bases the calculation of an execution plan on statistics that can be gathered with respect to index size, table size and other information. If such statistics are out of date, then the resulting execution plan may not be correct.
- The generation of an execution plan may also take time. Even though this is mitigated by repeating the test 100 times, the initially high value may skew the average obtained.

Further investigation into the issues identified is therefore required. This could take the form of repeating the tests on the existing platforms to re-validate the results obtained, removing the hints and allowing the optimiser to determine the execution plan. Running the tests on different software and/or hardware may provide further insights, as could redistribution of the impacted data and indexes onto different disks.

Workload tests also yielded an unexpected result whereby the *9-Intersection Pairs* tests were slower than the *Find Intersecting Objects* and *Find Objects with Relationship* tests. Tests should be re-run with all data removed from memory between tests to confirm the workload distribution results.

Although the graphs above do give preliminary trends both in terms of increasing numbers of users and in terms of increasing number of objects, extrapolating these to higher data volumes or higher numbers of concurrent users was not possible as in each case scalability tests were carried out for five different user-number combinations. Additional combinations, both at intermediate levels and at levels higher than the existing sets, are therefore required to validate the results obtained and the scalability trends observed.

Due to the 3D nature of the dataset, replication took place in all three directions, and each dataset was eight times as large as the previous one. This resulted in a difference of 900,000 objects between the two largest datasets. The creation of an additional dataset sized approximately 500,000 objects would lead to a more reliable prediction of time complexity of the underlying algorithms. Similarly, the creation of datasets containing 1.5 million objects and 2 million objects would confirm whether the performance reduction noted is due to the fact that the queries against the 1.08 million object dataset require disk reads whereas those against the 138,168 object dataset do not. If this is the case, similar performance results would be obtained for 1 million, 1.5 million and 2 million objects.

9.12 Extrapolating Results to a Real World Context

Although the main purpose of the tests was comparative, the results obtained for the 8-user 1.08 million object tests may also provide an indication of expected results for each structure from a real world system. However, the results obtained were captured through a series of dedicated tests, and may thus have been impacted by design decisions and the selected test environment. This impact is reviewed here.

9.12.1 Impact of the Test Environment

The performance tests described would ideally be run in a dedicated test environment, to ensure that there is no impact from other users on overall performance. However the absence of such an environment was mitigated by running three sets of tests. This shared environment implies that results for individual structures may reflect those expected for a similar sized real-world system, as a similar situation would be encountered, with the same database and database server being queried concurrently by multiple users and for many different purposes. Countering this is the fact that the test environment is read-only, which may yield artificially high performance times. Additionally, the execution times measured using the test harness do not take into account the time required to return the query results to the end user, or visualise these results, again leading to artificially high performance values.

9.12.2 Impact of the Test Execution Process

The execution of the comparative performance tests described, using an automated test harness to run simultaneous queries for multiple users, lead to indexes and data being pre-loaded into memory for some tests. Specifically, *9-Intersection Pairs* tests were executed first for each dataset size, pre-loading data for the other query types against the same dataset.

The use of randomly generated FEATURE_ID values for the *9-Intersection Pairs* tests should also be considered, as this resulted in many of the relationship tests returning DISJOINT. For STS and 3DFDS this requires the retrieval of two sets of primitives and then the INTERSECTION or MINUS operations on these primitives. Therefore execution time for DISJOINT relationships is similar to that for connected FEATUREs. For the Proxy for As-Required tests, the 3D R-Tree query only requires one read of the index no matter whether the FEATUREs are adjacent or not, as this will identify whether the second FEATURE_ID is on the same node of the index or not. However, for the *9-Intersection Pairs* tests, using random FEATURE_IDs against the MasterMap dataset meant that the computational geometry for the 'determine' relationship may not have been executed, leading to artificially rapid results.

9.12.3 Impact of the Artificial Dataset

The test dataset was generated by a process of replication, an approach that has been taken to ensure that all 9-Intersection relationships are represented. The objects represent simple objects, described by small numbers of coordinate tuples, and do not contain cavities or holes; no multi-part objects are included in the dataset, all surfaces are planar and all lines are straight. Three-dimensional objects have manifold surfaces and the dataset contains object pairs are also totally disjoint from other object pairs (in a real world dataset, each object would normally be

connected to more than one other object). The spatial extents of the dataset are also very small in the Y and Z directions.

9.12.3.1 STS and 3DFDS

For STS and 3DFDS the inclusion of complex and/or multi-part objects in the dataset would result in more shared primitives being returned for each query. This could negatively impact performance of the queries against a real world dataset, given that only 31% of the relationships modelled in the test dataset involve Body object types. However, this would only be the case where the real-world objects have a significantly higher number of associated primitives. Should objects be related to more than one other object (as would be the case in a real-world dataset) this would also impact performance of the *Find Intersecting Objects* and *Find Objects with Relationship* queries only if many more objects share the same set of primitives. The *9-Intersection Pairs* queries only reference two objects and performance would not be affected.

Spatial attributes (extent of the dataset, planar Faces, manifold or non-manifold structures, large numbers of coordinates used to describe the objects) do not impact STS or 3DFDS test results at all as structure-based queries do not access spatial elements of the data once the structures are populated. For example, a curved surface and a planar surface are both represented by a single entry in the TOPO_FACE table.

9.12.3.2 Proxy for As-Required

Performance of the coordinate geometry algorithms to determine the specific relationship between a pair of complex, multi-part objects could be much slower than that for simple objects considered, due to the additional coordinates. This may be particularly true if surfaces are non-planar, lines or curved, objects are described by many coordinates or objects are non-manifold. The small spatial extent of the test dataset may also lead to better than expected performance time for queries. An increased spatial extent would negatively impact both the size and performance of the R-Tree index created for the As-Required structure, as more index nodes would need to be created. Having more than two intersecting objects also results in an increased R-Tree index size, again reducing As-Required performance.

Table 76 below summarises the expected impact of a real world dataset on the performance results obtained.

Real-World Attribute	*STS*	*3DFDS*	*Proxy for As-Required*
Objects are highly connected (i.e. not pair wise)	Find Intersecting Objects and Find Objects with Relationship may be slower if dataset very highly connected.	Find Intersecting Objects and Find Objects with Relationship may be slower if dataset very highly connected.	Increased R-Tree index size may result in reduced performance.
Objects are complex (with holes and cavities)	No impact unless real world data very complex.	No impact unless real world data very complex.	Performance will deteriorate as coordinate geometry algorithms more complex.
Objects are multi part	No impact unless real world data has many parts per object.	No impact unless real world data has many parts per object.	Increased R-Tree index size may result in reduced performance.
Objects have curved surfaces	No impact – STS only references non-spatial topological primitives.	No impact – 3DFDS only references non-spatial topological primitives.	Performance will deteriorate as coordinate geometry algorithms more complex.
Objects have high number of coordinate tuples	No impact – STS only references non-spatial topological primitives.	No impact – 3DFDS only references non-spatial topological primitives.	Performance will deteriorate as coordinate geometry algorithms must deal with a higher number of input values.
Objects have non-manifold surfaces	No impact – STS only references non-spatial topological primitives.	No impact – 3DFDS only references non-spatial topological primitives.	Performance will deteriorate as coordinate geometry algorithms more complex.
Objects have large dataset extent	No impact – STS only references non-spatial topological primitives.	No impact – 3DFDS only references non-spatial topological primitives.	Increased R-Tree index size may result in reduced performance.
Structures use multi-segment *Edges*	Performance may improve due to fewer primitives.	Not applicable if Node/Edge relationship enforced. If this is not the case, performance will improve due to fewer primitives.	N/A

Table 76 - Impact of Artificial Datasets on Query Performance

9.12.4 Impact of Proxy Design

The selected Proxy was designed to estimate expected performance for As-Required topological queries against 3D datasets. Node-to-Face distance measurement was considered as forming the foundation of 3D relationship determination algorithms and the multiplication factor (described in Section 8.4.4) used to determine expected results for 3D relationship determination from the 2D case was based on the worst-case intersection algorithm complexity of $O(n^2)$. It is likely that this is not the case for the 2D implementation (although due to the proprietary nature of the Oracle software it may not be possible to determine this with any certainty). Similarly,

algorithms having this complexity will not be used in the 3D topology engine. Therefore the results obtained for these tests may be improved in practice, depending on the algorithm complexity of the existing 2D algorithm when compared to the implemented 3D algorithm.

To counter this improvement the 2D intersection-determination tests were executed for a single user situation only, and for a single dataset. Therefore no values for scalability or algorithm complexity for the 2D queries were obtained and hence factored in to the overall Proxy results. For the 3D element of these queries, minimal performance time was selected (0.5m index tolerance). However, in a real-world context the selected tolerance may be larger, and may depend not only on requirements of 3D topology but also on other queries such as distance measurement.

Taking all these factors into account, it is not possible at this stage to accurately predict the overall impact of proxy design on performance. Further work (described in Section 11.4.1) is required to develop a 3D topological engine and thus allow true performance comparisons to be made.

9.12.5 Impact of the Artificial Workload

An artificial workload was generated for the testing process, with each query type (*9-Intersection Pairs*, *Find Objects with Relationship* and *Find Intersecting Objects*) run in isolation of the others or of any additional queries end-users may executed. This was done to support comparative performance testing and also due to the fact that realistic workload information is not available. However, this approach could again lead to artificially low values for query execution time. In this case, the database is only executing one type of query at a time rather than determining multiple execution plans for many different queries as would be the case when topological queries are combined with metric, direction and attribute information.

Table 77 summarises the impact of the test environment on the query results.

Factor	*STS*	*3DFDS*	*Proxy for As-Required*	*Impact on Comparison*
Artificial Environment	Results artificially improved	Results artificially improved	Results artificially improved	None
Test Execution Process	*Find Intersecting Objects* and *Find Objects with Relationship* artificially improved.	*Find Intersecting Objects* and *Find Objects with Relationship* artificially improved	*Find Intersecting Objects* and *Find Objects with Relationship* artificially improved.	*9-Intersection Pairs* tests for Proxy artificially better.
Artificial Dataset	Little impact	Little impact	Results artificially improved	Proxy results artificially better
Proxy Design	N/A	N/A	Not possible to predict.	Not possible to make comparisons at this stage. Further work required.
Artificial Workload	Results artificially improved	Results artificially improved	Results artificially improved	None

Table 77 – Impact of the Test Environment

Allamaraju *et al.* (2000) suggest that 10 automated clients submitting queries at 1 query per second represent about 100 concurrent users (who would normally submit a query every 10 seconds). Given that overall query response time was generally of the order of milliseconds, leading to a new query being submitted approximately every 100 milliseconds, the eight simultaneous users thus could be taken to represent a total user base of 800.

Taking the above into account and reviewing the summary in Table 77, it can be seen that individual performance results obtained for STS and 3DFDS may be considered to be close to those attainable with a real-world dataset, unless this dataset is very highly connected or contains objects with many parts. However, the results obtained for the Proxy for As-Required tests may not reflect the situation with a real-world dataset, as execution time is closely linked to the nature of the objects being queried.

9.13 Selecting the Most Efficient Structure

Test	*Number of Users*	*STS (s)*	*3DFDS (s)*	*Proxy for As-Required (s)*
9-Intersection Pairs	8	0.34134	3.86793	0.386705
Find Intersecting Objects	8	0.17271	2.754576	0.314057
Find Objects with Relationship	8	0.18642	2.879998	0.768027

Table 78 – Summary Test Results, 1.08 million objects

Table 78 summarises test results obtained for 8-user, 1.08 million object tests. In the context of binary topological queries against the test dataset, STS provides the most efficient data structure when compared to 3DFDS, with binary relationship queries returning results approximately 10 times as fast. Although specific tests anomalies were identified for 3DFDS, these do not impact the overall comparisons.

Results also indicate that running 9-Intersection queries against STS gives more rapid performance than those against the Proxy. However, the multiplication factor (Section 8.4.4) represent an assumption of worst-case algorithm complexity meaning that actual results may yield faster performance once a topological engine has been developed. To counter this, the performance of the As-Required queries is impacted by the nature of the objects being queried, running identical tests against a real-world dataset would result in performance deterioration for these tests, whereas minimal impact would be observed for STS. Thus it is not possible to come to a definitive conclusion when comparing As-Required and STS approaches at this stage.

Depending on the implementation environment, other factors may also be relevant to structure selection – frequency of update, data quality and storage. If binary relationship queries are to be implemented in an environment where updates are frequent, the time taken to update the STS structure following changes in the core data may outweigh performance benefits to be gained from using this structure. This is particularly relevant if relatively few binary queries performed by users. However, data quality is an important consideration here. If the frequently updated As-Required data can be guaranteed to be of good quality without validation, then no further checks are required. However, if quality validation is required as part of the update procedure, the time required to validate the data for gaps, overshoots, undershoots and so forth must be taken into account for both cases. The time required to populate STS (a process which can make use of the primitives resulting from the quality control tests) may be minimal in comparison to validation time. Storage requirements may also guide the selection process, with STS requiring approximately 5500 MB additional storage for 1.08 Million Features when compared to the As-Required implementation (see Section 6.4.1).

The algorithms described and test dataset generated can be implemented within any Object-Relational database supporting spatial object types. Running similar tests in a different environment may yield different results, particularly for the comparisons between the STS and As-Required structures. Other considerations such as parallel processing of the R-Tree query (as described in Gorawski and Chechelski 2005) and further optimisation of the database may also improve performance, as would partitioning the FEATURE table.

9.14 Summary

This Chapter presented and reviewed the results obtained from the structure comparison tests described in Chapter 8. In the main, results met or exceeded the predictions given in Chapter 8, although a number of issues were encountered for the 3DFDS tests. These related to the embedding of specific index hints in the PL/SQL algorithms and whilst not impacting the comparatives or overall conclusions require further investigation.

The results clearly showed that, as expected, due to the reduced number of join queries, STS out-performed 3DFDS for all tests. Additionally, STS out-performed the Proxy for As-Required tests (although further testing is required to provide a more definitive conclusion and compensate for proxy design issues once a 3D Topological Engine is developed).

Taking the work described here and in previous Chapters, Chapter 10 investigates the possible commercialisation of this research, identifying potential end users, evaluating market size and listing threats and favourable trends.

10 Commercial Evaluation

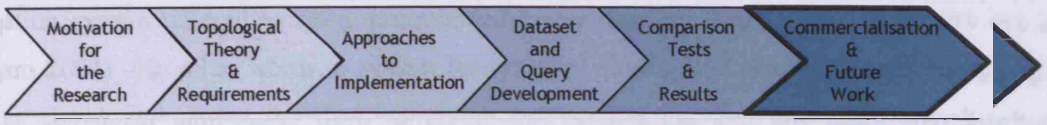


Figure 92 - Overview of Document Structure showing Context of this Chapter

10.1 Introduction

This Chapter details an initial examination into the commercialisation potential of the research, investigating potential partners and end users, including GIS software vendors and users of 2D GIS and 3D analytical tools[18].

The terminology employed here is deliberately non-technical to permit the distribution of the contents of this Chapter to a broader audience, although some familiarity with GIS and the concepts described in the remainder of this thesis is assumed. The approach taken towards the writing of this Chapter was similar to that for the rest of the thesis. However, rather than a review of academic literature, it involved assessment and review of commercial and market-related documentation and reports.

To this end, the Chapter commences with a definition of GIS and of 3D GIS, which is followed by a description of the product emerging from the research described in the remainder of this thesis. Market reports are used to provide an indication of the potential size of the 3D GIS market, and a review is carried out of the various tasks involved to identify where most financial value may be created and captured. This is followed by a brief assessment of favourable trends and risks facing the process of bringing a 3D GIS to market. The importance of promoting data and software integration by providing an open system is then discussed, and a preliminary plan of action outlined.

The primary aim of the Chapter is to suggest a strategy for achieving market dominance for 3D Topology and 3D GIS and to highlight potential issues and considerations that will need to be made along the way.

---

[18] This chapter is has been included in the thesis in fulfilment of the requirement for the Committee for Scientific Enterprise London (CSEL) thesis scholarship. As part of the PhD research process, CSEL funded attendance at a number of Masters in Business Administration second-year electives at the London Business School, as well as supporting the research described here. Material from these courses has been incorporated into this chapter, which deals with the potential commercialisation of the research.

10.2 Geographic Information Systems

A geographic information system (GIS) is traditionally defined as a "computer-based information system tailored to store, process and manipulate geospatial data" (Worboys and Duckham 2004) – in other words, a system for creating, storing and analyzing and managing data associated with some form of coordinate information and any associated attributes. Sources of such information include remote sensing, mapping, surveying and other technologies, and spatial technologies now assist in the processing of such data in the context of the organisation as a whole, integrating it with facilities management, billing, Enterprise Resource Planning, Computer Aided Design and other tools to provide enhanced decision support (Daratech, 2004).

Originally used by experts, academic and specialist users for specific tasks, 2D GIS has now become more mainstream (Parker 2004). This trend has been driven by greater availability of 2D data, the reduction in cost of hardware and software and an increasing understanding of the capabilities of GIS. The process of standardisation (such as the frameworks suggested by the Open Geospatial Consortium, OGC 2006) allows greater exchange of data, the emergence of web-based GIS including Web Mapping and Web Feature services has also encouraged the uptake of GIS. Research also shows that GIS is increasingly reliant on database technology, spatial analysis, topological modelling and graphics display (Daratech 2006a).

10.2.1 3D GIS

Key components of a 3D GIS have been identified as data capture, data analysis, data structuring and visualisation and the detection of spatial relationships (Zlatanova *et al.* 2002). Included under this umbrella are methods such as the automatic generation of terrain models from aerial imagery or laser scanning, the semi-automatic reconstruction of 3D objects such as buildings and the display of object textures. Visualisation of 3D data is now available for both desktop and web-based systems, and research into 3D GIS analysis has been carried out by authors including Zlatanova (2000), Pfund (2001) and Wang and Gruen (1999).

In short, requirements for 3D GIS can be said to be an extension of those for 2D GIS, along with appropriate tools to visualise, analyse and explore 3D datasets. Van Oosterom *et al.* (1994) describe functionality such as visibility diagrams, conversion of TINs to contours, 3D generalization, watershed runoff and drainage basins and visualising the internal structure of objects as examples of these analysis tools.

10.3 Current Status of 3D GIS

Although 3D GIS are marketed by the mainstream GIS vendors, they do not offer full capabilities corresponding to those in 2D. In fact, many do not handle true 3D objects, but rather provide functionality to visualise and manipulate 2.5D surfaces. A brief review of the 3D capabilities of four mainstream software products illustrates this point (note that this review is not intended to provide a description of each product as a whole).

Environmental Systems Research Institute – ArcGIS (ESRI 2006). ArcGIS provides a 3D Analyst toolkit which allows visualisation and navigation through 2.5D surface data. 3D Analyst can generate its own TIN surfaces but cannot generate TIN envelopes that completely encompass a solid. It allows users to query a surface, determine what is visible from a chosen location, view perspectives, drape images over a surface, manipulate 2.5D data and run visibility analysis, surface analysis and terrain analysis.

Intergraph Geomedia Professional and MGE Terrain Analyst (Intergraph 2006). The Terrain Analyst tool provides capabilities to create, manipulate, display, and analyze digital terrain models that can be represented as triangulated irregular networks (TIN) or regularly spaced matrices (grid). In both cases, this represents the processing of 2.5D data rather than true 3D data. Terrain Analyst also permits connection to a relational database. Similar functionality is incorporated directly into Intergraph's GIS product, Geomedia Professional.

MapInfo Professional (MapInfo 2006). Support for 3D is limited to the '3D Data storage and manipulation tool' which allows users to store and manipulate 3D data in Oracle Spatial (Oracle 2006a) but not to visualize the results in 3D. MapInfo also offers 3D mapping capability for grid point based maps.

Autodesk Map 3D (Autodesk 2006). This package provides surface visualization, and also allows the creation of links from 2D mapping data to associated 3D CAD drawings. Overlay and analysis of multiple CAD drawings is included in the functionality provided. Topology is offered in the form of Network and Planar topology.

The fact that mainstream GIS vendors offer elements of 3D support is an indication of the growing interest in this functionality by end users. However, a number of issues must be addressed before a 3D GIS can be released as a mainstream software product. These challenges (listed by Smith 2006) are both technical and organisational, and are summarised here:

- 3D data is of variable quality, and often originates from purely visualisation-based applications. As such, object attribution and data quality assurance processes are required before such data is incorporated within a 3D GIS.
- Moving from 2D to 3D is as challenging as moving from paper maps to onscreen maps. In particular, issues relating to 3D visualisation and mobile GIS are of importance. In an ideal world, it would be possible to superimpose the 3D data on the real world. 3D GIS itself also presents another learning curve for end users.
- Although a number of 3D applications have been identified, particularly within the fields of urban planning and emergency management, the lack of true 3D GIS is itself currently impeding further spread of the technology. Users lack understanding of the potential of such systems and how they could be implemented using their specific datasets.
- Due to the lack of demand, investment in 3D GIS by vendors is currently low. The GIS community needs to provide examples of working systems and engage end-users with the technology.

Encouragingly, the number of non-commercial data structures for dealing with 3D data is relatively high (Molenaar 1990, Zlatanova 2000, Pilouk 1996, Coors 2003, Pfund 2001, De La Losa and Cervelle 1999, Ladner *et al.* 2001), although these have primarily been developed for specific applications or application domains. However, 3D GIS are yet to undergo the consolidation and mainstream acceptance processes required to ensure wider distribution of the functionality. In particular, topological analysis of 3D data is not available in any of the commercial products listed above

10.4 The Product – 3D Topology Toolkit

The first important aspect of evaluating the commercial potential of this research is the provision of a clear definition of exactly what is to be commercialised. This in turn will allow the identification of any further work required to move the research from its current state towards something that could be employed by end users.

Two deliverables can be identified from the research – a data structure that provides high-performance 3D binary topological queries, and a series of PL/SQL routines that implement the required queries, mapping user terminology to defined relationships. Both of these depend on the presence of an Object-Relational database within which to store the spatial data. It is anticipated that the Simplified Topological Structure and the associated PL/SQL routines described in this thesis will constitute the foundations of the product.

However, one key component fundamental to the implementation of topology in 3D GIS has yet to be developed – namely the 3D topological engine required to populate and maintain data within the structure. Functionality to be performed by the Engine includes data quality control algorithms (some of which are described in Appendix 1) and the primitive identification and

database population (examples of algorithms and approaches that could be applied to this task are given in Section 4.3). The development of such algorithms was outside the scope of the current research – however, it may be possible to identify commercial products containing the required tools. Parasolid (2007) provide one potential tool, the Bodyshop, which allows the creation of valid solid models and performs validation and cleaning of topology. Editing tools are also provided which include patching of holes in 3D models and validation of topology and geometry. Other libraries such as CGAL (Computational Geometry Algorithms Library, 2007) may also be relevant. It will be assumed for the purposes of product definition that such an engine is available, leading to a product that may be defined as a 3D Topology Toolkit (Figure 93). Note that it is assumed at this stage that the 3D Topology Engine will be closely coupled with the database. Tighter integration of database and engine will also be considered if required for performance.

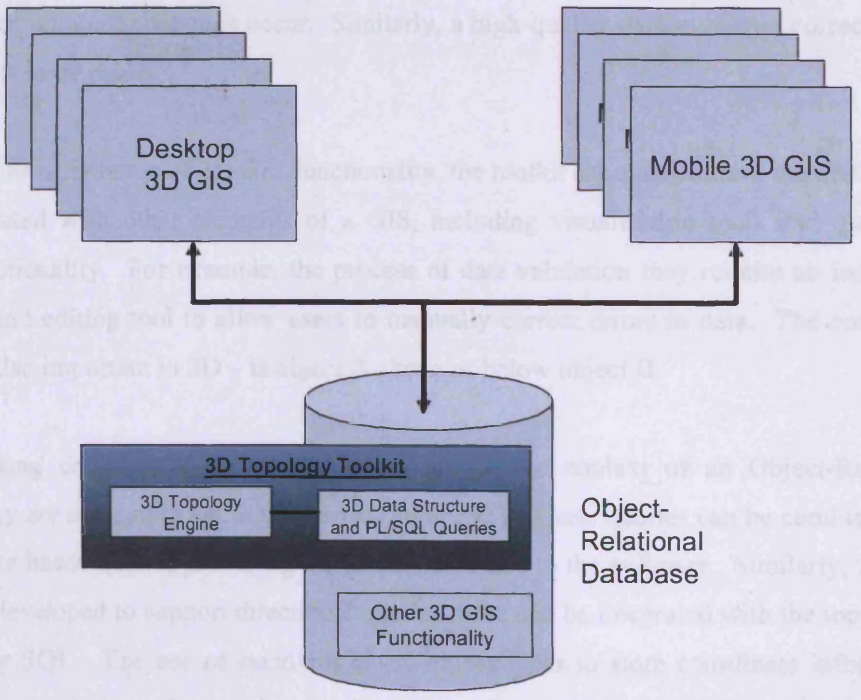


Figure 93 - Architecture of the 3D Topological Toolkit

10.4.1 Features of the Toolkit

Having defined the product at a high level, it is now possible to summarise required features. These relate both to the functionality to be provided and to the means of providing access to this functionality, and include:

- Data validation and quality control (ensuring that there is no missing data and that objects do not overlap incorrectly).
- The determination of topological relationships between two 3D objects (for example is object A next to object B, inside object B, partially inside object B?).
- The identification of any objects having a particular topological relationship with the object in question (which buildings are next to this one?).
- The creation of networks through 3D objects, in particular building models, to allow emergency escape routing and disaster management planning.
- Modelling of both curved and planar 3D surfaces.
- Modelling of true 3D objects.

One of the key consequences of implementing 3D topology using a data structure such as those described in this thesis is the possibility to provide data quality assurance processes as part of the structure population process. Having correctly structured data provides a number of benefits, including the ability to run network routing algorithms against the dataset to define possible escape routes to handle emergency situations. Generating an automated accurate network representation of urban structures is not possible without a corrected 3D dataset where no overlaps or unintentional gaps occur. Similarly, a high-quality dataset allows correct metric measurements to be made.

To support a broader range of 3D GIS functionality, the toolkit should also have the ability to be easily integrated with other elements of a GIS, including visualisation tools and query and analysis functionality. For example, the process of data validation may require an interactive 3D display and editing tool to allow users to manually correct errors in data. The concept of direction is also important in 3D – is object A above or below object B.

As the existing components have been developed in the context of an Object-Relational database, they are accessible via SQL queries (Figure 93). These queries can be combined with other attribute based queries providing a seamless interface to the end-user. Similarly, should a package be developed to support directional queries these can be integrated with the topological queries using SQL. The use of standards-based object types to store coordinate information, opens up the possibility of accessing functionality using multiple 3D GIS applications. The toolkit can be deployed within any Object-Relational database supporting spatial object types.

It is envisaged that the toolkit will primarily be deployed within the context of a 3D GIS, integrating with other functionality such as data editing, visualisation, metric and directional queries. The interoperable toolkit could be integrated as one of a series of components, each optimised to support a specific element of 3D GIS functionality. In the short term, it may be

possible in the shorter term to deploy the toolkit as a stand-alone product to existing users of 2D GIS who wish to perform topological queries against 3D data sets but may not at this point be interested in other 3D GIS functionality.

10.4.2 Complementary Assets and Products

Complementary assets can be defined as those that aid and enhance the use of the core product. In the case of the 3D Topology toolkit, this relates primarily to tools which allow users to visualise 3D data, edit the data to correct any errors identified by the topology engine and display the results of binary or network 3D topological queries. These products are required to allow end-users to benefit from the full functionality offered by the toolkit. Other components of a 3D GIS can also be considered. These include tools to execute metric, directional and non-spatial queries against the data.

Taking a broader view of 3D GIS, complementary products also include 3D datasets. In particular, 3D topographic mapping can be employed as a back-drop to many other datasets in a similar manner to current 2D topographic mapping. Such 3D data, if appropriate detail is provided, can also be utilised to underpin the capture of other datasets – for example, 3D cadastral models.

In conjunction with data provision, GIS services and consultancy can also be considered as complementary. Configuration of the topology engine to specific end-user requirements may be relevant. Support including the identification of areas where 3D GIS may benefit an organisation, and the provision of systems integration services are also complementary to the development of 3D GIS software.

10.5 Sizing the GIS Software Market

In order to identify the financial potential of the 3D Topology Toolkit, it is useful to first obtain an idea of the size of the GIS Software market as a whole. Daratech (2006b) predict this to be $1.77 billion US dollars worldwide, split as shown in Table 79.

Sector	*%*	*US$ Million in 2007*	*Growth 2004-2006*
Public Market	39%	690	15%
Regulated	38%	665	7%
Private Sector	23%	415	Not Known

Table 79 – GIS Software Market Revenue 2006, Daratech (2006b)

10.5.1 Market Sizing for 3D GIS

The current 2D GIS software market is well established. The market has undergone consolidation within the last ten years, leading to ESRI (ESRI 2006), Intergraph (Intergraph

2006), GE Energy (GE Energy 2006), Autodesk (Autodesk 2006) and MapInfo (MapInfo 2006) having a total of 68% of the market share (assuming that figures published by Daratech 2002 can still be applied). The emergence of Object-Relational databases has driven the move from a stand-alone system towards the integration of GIS with corporate modelling, decision support and asset management tools (Mannings and Parker 2006). Standards such as those defined by the Open Geospatial Consortium (OGC 2006) have fostered data exchange and integration. Web-based GIS and location based services move GIS from specialist to non-specialist users and encourage growth in the industry.

However, the situation with regard to 3D GIS is much less clearly defined, and in fact may be compared to the status of 2D GIS fifteen or twenty years ago. Within the market sizing figures given above, there has been no differentiation between 2D and 3D GIS, primarily due to the fact that commercial 3D GIS do not currently exist. However, it can be assumed that, once fully functional 3D systems are available, they may gain an increasing percentage of GIS software revenue. Hypothetical calculations are given here for 5%, 10% and 15% of revenue in 2006 (Table 80).

These figures represent the potential market for the total 3D GIS product. Given that 3D Topology provides core functionality along with visualisation, total revenue for such functionality may be assumed to generate a maximum 50% of the 3D market, if it is assumed that data cleaning functionality is included here. Again, there is no research to underpin this assumption, so an alternate figure of 10% should also be considered. The resulting potential market sizing is shown in Table 80 below:

	5% (US$)	*10% (US$)*	*15% (US$)*
3D GIS Software Sales	88.5 million	177 million	265.5 million
3D Topology Software Sales – 50%	44.25 million	88.5 million	132.75 million
3D Topology Software Sales – 10%	8.85 million	17.7 million	26.55 million

Table 80 – Hypothetical Market Size for 3D GIS

Table 80 shows optimistic values for total potential market size, assuming that the future GIS software market retains its current size, that 3D GIS functionality captures a reasonable share of this market in a very short space of time and that the 3D Topology Toolkit captures the entire worldwide market for 3D Topological functionality. It does not, however, take into account potential revenue from data or services, as discussed when considering the broader GIS value chain below.

10.6 *Creating and Capturing Value*

Figure 94 – Generating Revenue (Masini 2006)

Figure 94 provides a framework to assess the likelihood of a particular technology product generating sustainable revenue. Uniqueness in this context refers to the possibility of others developing similar products. Examining the 3D Topology Toolkit, uniqueness of the data structure or the PL/SQL code described elsewhere in this thesis is difficult to maintain. However, the uniqueness of the algorithms required to build 3D Topological Engine to populate the structure and maintain data can be protected, as significant effort is required to develop such functionality. Given the open design overall, and the use of SQL to ensure that multiple 3D GIS products can make use of the functionality provided, complementary assets are freely available. This places the 3D Topology Toolkit in the top left quadrant of the above figure.

However, by itself, the Topological Engine will only generate revenue through licensing. It is therefore important to consider other 3D GIS technology and associated services in order to identify additional areas where value can be created.

Figure 95 - GIS Value Chain

Figure 95 above shows a high-level GIS value chain, which lists the activities of an organisation that can be directly related to revenue generation. Technology Components represent software packages such as the 3D Topology Toolkit or a 3D visualisation tool and the Core GIS Technology represents an integrated software package containing these components. Data and Data Quality represent any processes concerned with the provision of accurate data to end-users, be it capture, value added analysis or quality control. Integration services are perhaps the most

important component of the chain, and represent the process of integrating GIS and associated data with other corporate systems such as Enterprise Resource Planning, Facilities Management and other decision support tools. Finally, Support represents the ongoing maintenance and issue-resolution tasks required for any software installation. Two types of vendors can be identified along this chain – the specialist vendors, who provide software components within a GIS or who focus purely on the services or data components of the industry, and the integrated vendors, who provide services all along the value chain, perhaps in conjunction with specialist partners.

Figure 96 - Overall GIS Revenue by Sector (Daratech 2006c)

Figure 96 shows the proportional split for world-wide GIS revenue. As can be seen, whilst GIS software sales (the first two components of the value chain in Figure 95) represent 50% of this revenue, a significant proportion is also allocated to data and to services in general. Thus, considering the value chain above, it is the vendors that provide an integrated solution, from software development through to services and support, who are likely to create and capture more value across the chain.

As data quality improvement results from the process of populating a topological data structure, offering services relating to 3D data quality improvement may be one area where value could be captured, given that much 3D data is currently held in formats optimized for visualization which may not enforce rules such as 'adjacent buildings should not overlap' and 'all buildings should be represented as correctly closed polyhedra'.

10.6.1 Target Partners

The 3D Topological Toolkit proposed here resides within an Object-Relational database, and provides direct access to its functionality by means of SQL queries. A user interface will be required to allow end-users to configure the process of data migration and structure population. Thus working with Object-Relational database vendors to ensure that the most efficient performance is obtained from the Toolkit may be appropriate.

Within the context of GIS, overall revenue is split between the service end of the value chain and the software development end. The 3D Topological Toolkit described here has been designed to allow integration with many 3D GIS client software packages. Therefore, it may be appropriate to consider partnership with one or more existing GIS software and service providers to ensure that the Toolkit is tightly integrated with the 3D GIS. Time to market is important here – first-mover advantage (i.e. being the first 3D Topological toolkit to market) increases the possibility of market dominance. Similarly, product branding may play a part in this process. Partnerships with data providers should also be considered, given their interest in improving the quality of their 3D data, and thus widening its potential areas of application.

10.6.2 Target Markets

Target market segments for 3D GIS and 3D Topology include ecological studies, urban mapping, environment monitoring, landscape planning, geological analysis, architecture, civil engineering, mining exploration and archaeology (Ellul and Haklay 2006). Crisis management is also becoming increasingly important, both due to natural disasters and terrorist attacks.

Three-dimensional GIS and 3D topology offer new methods of handling, querying and analysing data to database users. It is suggested that a small number of target users of 3D GIS are selected for first implementation, before moving to a more generic product rollout offering 3D GIS as a commercial-off-the-shelf product. Identifying a number of potential users in the Urban Modelling and Earth Science markets would create a number of early adopter sites where the full potential of 3D GIS could be evaluated and the product stability established. Although the specific size of each of these target markets is unavailable, they fall into the general category of Public Sector, which constitutes 39% of the overall GIS software market. Assuming that this figure remains constant for 3D GIS and that Cadastral and Earth Sciences form approximately 50% of the Public Sector proportion of this revenue, it can be predicted that the maximum initial annual market size for 3D GIS as a whole will initially be $US 6.64 million (based on the 5% figure in Table 80). This could then be followed on by full product rollout of the 3D GIS as a

whole, targeting a wider range of markets including the utilities, telecommunications and transportation.

10.7 Favourable Trends

Trends favouring the growth of GIS in general and 3D GIS in particular include the availability of increasing computer power at lower cost, the lowering of GIS package prices, and the increasing availability of data (Frost and Sullivan 1998). GIS is no longer the preserve of academics or highly trained specialists using expensive equipment and geographic information has become a mainstream information type, with extensions beyond 2D to 2.5D, 3D and 4D (time) (Mannings and Parker 2006). The incorporation of GPS into mobile devices makes location based services more important, and additional sources of 3D data are emerging, fuelled in part by the availability of cheaper GPS receivers. Additionally, security and anti-terrorism initiatives result in a demand for technical support, and a consequent need to upgrade the technology used by public service agencies to support emergency situations. Daratech research indicates that 10% increase in governmental spending at all levels of GIS technologies, including hardware such as handheld devices – much of this is to support security initiatives (Daratech 2005).

Furthermore, a trend towards enterprise GIS can be identified, with the emergence of systems integrated with billing and customer relationship management amongst others. GIS is no-longer stand-alone. Integration allows more efficient usage and sharing of data and resources and also improves decision quality while lowering maintenance and support costs.

A number of initiatives and trends favour the increasing availability of 3D data. In particular, a high number of initiatives in 3D city modelling (including in particular CityGML, CityGML 2006) can be identified. Lapierre (2007) notes that both GoogleEarth and LocalLive/Virtual Earth are currently investing in the creation and delivery of 3D city models, and suggests that this is the sector of the Geospatial industry that is likely to see most growth in 2007.

Other initiatives supporting the growth of city modelling include the Open Geospatial Consortium's Web Services Phase 4 initiative, which includes the establishment of specifications relating to CAD/GIS/BIM (Buildings Information Models). Additionally, Oracle (Oracle 2007) plan to include 3D spatial data types in their next release (11g). It is therefore likely that any initial data gap will be initially be filled in specific sectors such as urban modelling and geological and Earth Science. The development of a generic 3D Topology Toolkit, which will support analysis of data from multiple sources, will ensure that the range of available 3D data can be utilised.

10.8 Threats

A number of unfavourable trends and issues can be identified with respect to the development and deployment of 3D GIS. Firstly, there is a lack of interoperable standards for 3D data. 2D GIS employ simpler data structures resulting in reduced storage space. Data output can also be easily plotted on existing printers. Lack of similar functionality in 3D GIS may deter existing 2D GIS users from migrating to 3D.

There is minimal awareness of the functionality offered by 3D GIS and potentially high cost of initial 3D GIS and data (issues listed by Frost and Sullivan 1998 in the context of 2D GIS, but equally as relevant now in the 3D context). Additionally, Goldstein (2003) points out that the GIS industry is driven by technology rather than by the requirements of end-users. In the 2D context, 80% of application tools provided will not actually be used. Lack of integration with corporate systems is also an issue.

The risks associated with the development of a 3D Topological engine should also be considered. These relate particularly to engine performance during data creation and maintenance operations.

A number of these threats can be mitigated by the development of standards-based tools and involvement in the overall standards definition processes. Additionally, demonstrations, close partnerships with 2D and 3D GIS software vendors and with potential end users, and general increased of awareness of the technology through conferences and advertising will also help to overcome the issues described.

Given the similarity between issues currently faced by 3D GIS and those previously faced by 2D GIS, it may be possible to apply a number of the lessons learned in the 2D context in early mitigation of these issues in. In particular, the importance of interoperability has now been identified as key to the success of GIS. This is discussed further below (Section 10.9).

10.8.1 Competing Products

Given that true 3D GIS software has not yet reached the mainstream market, competition to 3D GIS software takes two forms. Firstly existing 2D GIS, where methods have been developed to handle problems which are intrinsically 3D. For example the GE Smallworld (GE Energy 2006) system offers a system of multiple 'worlds' whereby users can visualise the 2D cross-section of a utility trench, linking through from the 2D linear representation of the trench. Similarly, Autodesk Map 3D (Autodesk 2006) allows users to navigate from 2D maps to 3D

architectural CAD drawings for various objects on the map. Due to familiarity, GIS users may consider these solutions adequate for their needs, removing the need for migration to true 3D GIS. Secondly, specialist 3D visualisation and analysis software, such as Advanced Visualisation Systems (AVS 2006) and IDL (IDL 2006) should also be considered. Tools such as these offer data visualisation of 3D scientific and engineering datasets and provide a toolkit to allow software developers to write custom tools, including visual analytics for business. Many such tools also provide links and query functionality for relational database data.

Additionally, it is possible that rival GIS vendors may also invest in the development of such technology. Technology to perform As-Required determination of topological relationships represents a particular threat, if such relationships can be identified rapidly and queries can be run against data stored in an Object-Relational setting.

Figure 97 - Types of Innovation (Henderson and Clark 1990)

Figure 97 depicts a framework for the identification of different types of innovation. In this context, architectural innovation refers to the reconfiguration of an existing system to link established components in a new way. Modular innovation maintains existing functionality but replaces tools with more modern technology. Radical innovation establishes a new dominant design and incremental innovation refines an established design. Identification of the type of innovation a technology provides may give insight into the reaction of competitors.

Using the above definitions, 3D GIS falls into the category of an incremental innovation. Given this, and the fact that the impact of 3D GIS technology on the market will be gradual rather than immediate and complements existing 2D GIS (i.e. is sustaining rather than disruptive) competitors are likely to detect the emergence and will thus react to this technology and the

trend towards 3D GIS in a timely manner. Given the existing trend to badge products with a 3D name when true 3D data is not handled, it can be assumed that GIS software vendors have already recognised these trends.

10.9 The Importance of Interoperability

A key issue facing the 2D GIS community was, and still remains, the issue of interoperability – non-interoperable systems do not work together, resulting in issues when sharing data and computing resources and when integrating application software from various vendors. This in turn results in additional expense to integrate such systems, often through the deployment of further integration software. It increases technology risk, resulting in reduced benefits from the deployment of technology, as well as real-world risks if information is not communicated in a correct and timely fashion. Considering the number of different software packages handling spatial data, ranging from Object-Relational databases to CAD, Location-Based-Services, facilities management, surveying and earth imaging further highlights the importance of interoperable systems (Reichard 2004).

The Open Geospatial Consortium is "a non-profit, international, voluntary consensus standards organization that is leading the development of standards for geospatial and location based services" (OGC 2006) and defines an open standard as one that:

- Is created in an inclusive, international, participatory industry process.
- Is owned in common.
- Has free rights of distribution.
- Does not discriminate against persons or groups (e.g. due to cost).
- Is technology neutral.

Of particular relevance to this research are the emerging Web 3D Service (W3DS) and CityGML standards (Kolbe 2005, OGC 2006, CityGML 2006). Kolbe notes that 3D city models constitute an important source for of information for emergency planning and response allowing the development and testing of various scenarios and supporting the disaster recovery planning processes. To be of benefit to such planning processes, data from various sources must be integrated. The W3DS standard has thus been proposed to facilitate this task, particularly in the context of integrating online data from varying sources prior to rendering it using a 3D graphics engine.

CityGML addresses a higher level of interoperability – at data model level, providing an application schema for storage and exchange of virtual 3D city models. Rather than focussing on the integration and visualisation of 3D spatial data, CityGML aims to provide common definitions for basic entities found within a 3D city, providing a neutral schema to allow users of such data to seamlessly move from one application to another.

Given the issues generated by non-interoperable systems, and the fact that the overall system architecture of the 3D Topological Toolkit has been designed to facilitate integration with multiple 3D GIS, it is important that toolkit developers are involved in the standards setting processes for both W3DS and CityGML, and that where possible compliance with the frameworks suggested by the OGC is achieved.

10.10 Preliminary Plan

10.10.1 Developing a 3D GIS

The 3D Topology engine provides the missing piece in the jigsaw of the 3D Topology Toolkit. It is only once this has been developed and tested that the full potential of such a toolkit can be evaluated. The toolkit can then be integrated with additional 3D GIS database tools and support for directional and metric queries, perhaps developed by third-party partners. This will provide an open interface to partners who may wish to provide the required user interface, including visualisation, query and editing tools.

10.10.2 Market Research

In the context of the research described in this thesis, a number of areas requiring further investigation can also be identified. Currently, GIS Market Reports such as those provide by Frost and Sullivan (1998) and Daratech (2005) do not focus specifically on 3D GIS, as commercial products do not yet exist. Therefore, a survey of candidate users, accompanied by demonstrations of available functionality, may provide a more accurate estimate of potential market size and hence revenue. Such a survey also provides a means to prioritise functionality developed and identify early adopters. This, along with toolkit development, forms the next steps in the development of the 3D Topological Toolkit.

10.10.3 Moving the GIS market from 2D to 3D

Figure 98 below shows the stages that a technology product passes through to achieve market dominance. Given the current lack of a 3D topological engine, the 3D Topology Toolkit can be located between T0 and TP on the diagram – i.e. it is undergoing Research and Development Build Up. Some technical feasibility testing (Phase 2) has also been carried out in the form of a data structure and binary topological queries proposed by this research.

Figure 98 - Milestones towards Product Dominance[19]

Taking the favourable trends into account along with the proposed target market (initially urban and Earth Science applications) it is envisaged that 3D GIS will gradually impact the GIS market, rather than having a radical, immediate effect. It can be predicted that 3D GIS technology will co-existing alongside existing 2D systems for some time into the future. Similar growth will be seen when complementary products are considered.

To encourage this process, it is important that 3D GIS vendors proactively market the software to allow end-users to gain an understanding of its potential. Marketing strategies (including demonstrations, conferences, advertising, free downloads) should be targeted not only at existing GIS users but also at other users of 3D data.

Eventual market dominance will depend heavily on 3D GIS vendors overcoming the threats and taking advantage of the favourable trends highlighted. They may need to consider providing free or low-cost access to toolkits and software to achieve market dominance. This may be more feasible for the integrated vendors who are able to cover costs through the provision of data quality control services and as systems integrators, leading to the likelihood that these types of organisations will eventually become market leaders.

10.11 Longer Term Possibilities

Given the investment required to create a 3D topological engine and integrate this with other 3D GIS components, it is unlikely that very significant amounts of revenue can be captured if the applications of 3D Topology are confined to 3D GIS, although licensing and data quality activities will contribute somewhat. Other revenue-generation ideas must therefore also be considered. Specifically, other areas where 3D data is also manipulated and used could be of significance here.

The population of the 3D topological data structure results in high quality 3D data, stored within an easily accessible format. 3D R-Tree queries can be used to rapidly retrieve data specific to a

[19] Technology Strategy Course Notes, Andrea Masini, April 2006

particular area of interest. More importantly, storing the data in an Object-Relational database means that it can be easily attributed. Investigating the application of attributed 3D data virtual reality or gaming may therefore be worthwhile – can an Object-Relational database provide data to a visualisation engine rapidly enough to meet requirements? What, if any, value does attributed data add to the end-user's experience of the game and to the overall game design?

Worboys and Duckham (2004, pg. 304) note the use of directional topology to support the 2D generalisation process (i.e. to provide data suitable for display at different levels of detail). Again, this concept could be extended to 3D, and in particular to urban datasets, where it is important that a building is maintained on the correct side of a road through different levels of detail required for display, even though the building itself may be merged with others.

10.12 Summary

This Chapter examines issues relating to the commercialisation of the research described in the remainder of this thesis. A number of factors require consideration in this context. Firstly, the importance of developing a fully functional 3D topological engine to complement the work described in this thesis has been identified. This can then be integrated with multiple 3D GIS software packages as these become available, adding topological functionality to the 3D GIS product suite. The development of 3D GIS in general must thus keep pace with the development of the topology engine. Partnership with GIS software vendors may help to achieve this.

Sources of potential revenue include not only the core product (a 3D Topological Toolkit) but also the provision of 3D data and services such as integration of 3D GIS with corporate software. Given the lack of existing 3D GIS, measures must also be taken to promote uptake of such software, and to ensure that end-users fully understand the scope and potential of the tools offered. Finally, it is important to participate in the standardisation process which will in turn encourage further dissemination of 3D topology and 3D GIS.

Taking this information, along with the results of the requirements review and performance tests, into account, the final Chapter of this thesis summarises the work undertaken, linking back to the research questions defined in Chapter 1. Recommendations for further research are also presented.

11 Research Summary and Further Work

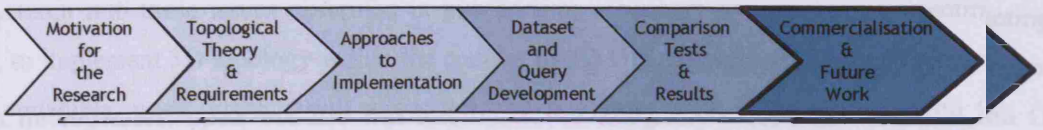


Figure 99 - Overview of Document Structure showing Context of this Chapter

This concluding Chapter of this thesis reconsiders the research questions and outlines how they have been addressed, highlighting areas where this work differs from previous approaches. This is followed by a review of further work to be undertaken, in terms of enhancing and refining the processes described as well as in terms of integrating topology in the wider context of 3D GIS.

The requirement for the research described in this thesis can be identified through a review of the current status of GIS and of 3D GIS in particular. Three-dimensional applications are becoming more widespread as hardware power increases and software is developed to take advantage of this fact. 3D applications implementing visualisation tools are now widely identified. 3D virtual reality environments and games are also common place. However, many of these applications are based around proprietary systems and data structures. Analysis-based applications underpinned by database information are not yet commonplace.

Databases are generally increasing in size and supporting more and more users. Object-Relational databases are available, and SQL has moved from a basic query language to include adaptations for querying multi-media data. Table and index partitioning is available to take advantage of increased performance obtained by storing data on multiple disks and thus having parallel disk read processes. Parallel execution of code is also possible, particularly on multi-processor architectures. The requirements for 3D GIS are also increasing across multiple domains. More and more, GIS tends to be integrated with other information sources and with corporate systems from customer management to billing and finance. Cadastral systems, earth science applications and emergency planning have been identified as potential application domains.

Countering this, many so-called 3D GIS support 2.5D data (representing a surface - created as each coordinate pair can be assigned at most one height value) rather than true 3D data, and 3D metric and directional queries, visualisation, navigation and interrogation of 3D data through the GIS interface are not yet implemented. Standard structures and functionality for the storage and manipulation of 3D data in a GIS have not yet been agreed. Requirements for 3D topology (long

been identified as an important component of a GIS) are not clearly defined and functionality has not yet been implemented commercially.

The research into these issues described in this document furthers the process of selecting a system to implement 3D topology within the context of 3D GIS. Developing such as system in an interoperable, open, manner will allow it to be integrated with other best-of-breed GIS components.

11.1 Addressing the Research Questions

Four research questions have been addressed by the work described in this thesis. The first part of this research (Chapter 2 and Chapter 3) answers the questions:

Which areas of GIS applications will benefit most from efficient handling of topology in 3D? Which binary topological relationships should be modelled to generate this benefit?

A list of cross-disciplinary high level requirements for topological functionality in 3D GIS was derived through a review of literature related to applications utilising 3D data and by examining current uses of 2D topology. The main focus of this work, however, is the requirement for and implementation of binary topological relationships as these underpin higher order functionality. It was concluded that that three generic topological queries are required – namely:

- *Find Intersecting Objects.*
- *Find Objects having a specific Relationship with this one - abbreviated to Find Objects with Relationship.*
- *Find 9-Intersection Relationship between pairs of objects – abbreviated to 9-Intersection Pairs*

In parallel with this process, following a review of existing 3D topological frameworks, the 9-Intersection framework was selected as the most appropriate to support implementation. This was due to existing approaches to handling compound and complex objects, and also to the methodology to group the numerous relationships. Building on this, a mechanism was developed to allow end-users to customise the cross-referencing between the three-digit R-Value defining each relationship and their discipline-specific interpretation of the relationships within the context of a generic system.

The second research question (answered in Chapters 4 and 5) asked:

How can these relationships be implemented in an Object-Relational database context?

A review of existing approaches and data structures underpinning the implementation of binary 3D topological queries led to the selection of two of these for further comparison. 3DFDS was selected as representative of existing 3D topological data structures. This structure was extended to handle multi-part objects and to represent the interior of a 3D object with a Volume primitive. Along with this, in the absence of a 3D topological engine, Oracle's spatial object

(SDO_GEOMETRY) was used to underpin a Proxy for As-Required queries. A third data structure – STS – was developed to validate the hypothesis that query performance is reduced by the number of join queries and exception tables required in a traditional B-Rep structure.

Thirdly, the thesis answers the following question:

> *Of the various implementation options, which approach provides more efficient performance results?*

A test dataset (Chapter 6) was created to represent binary relationships between simple objects in 3D, and then replicated to 1.08 million objects. Overall storage requirements for STS totalled 6207 MB, for 3DFDS 7083 MB. For the As-Required approach, having an index tolerance of 0.5 m, total storage required is 496 MB.

Oracle SDO_FILTER queries utilising a 3D R-Tree index were used with the As-Required structure to implement a Proxy for the topology engine, with the addition of 2D query results to improve this Proxy. Set-based algorithms were developed to implement the queries against STS and 3DFDS. The resulting extended SQL queries implemented to meet the requirements for 3D binary topological relationship are summarised in Table 81 (both As-Required and structure-based queries are described in Chapter 7).

Requirement	*Implementation*
Identify adjacency of the different polyhedra in a model, and also between combinations of 0, 1, 2 and 3D objects and between complex objects.	PL/SQL query of 3DFDS or STS. Query takes the form: `SELECT * FROM TABLE( TOPOLOGY.QUERY('ADJACENT',<< FEATURE ID>>));`
Identify intersection between 3D, 2D, 1D, 0D combinations.	PL/SQL query of 3DFDS or STS. Query takes the form: `SELECT * FROM TABLE( TOPOLOGY.QUERY('INTERSECT',<< FEATURE ID>>));`
Identify containment of geometries of 3D, 2D, 1D and 0D objects and also of complex geometrical objects.	PL/SQL query of 3DFDS or STS. Query takes the form: `SELECT * FROM TABLE( TOPOLOGY.QUERY('CONTAINED',<< FEATURE ID>>));`
Identify disconnectedness of two objects.	PL/SQL query of 3DFDS or STS. Query takes the form: `SELECT * FROM TABLE( TOPOLOGY.QUERY('DISCONNECTED',<< FEATURE ID>>));`
Identify the specific topological relationship between two objects (3D, 2D, 1D, 0D combinations) and also of complex geometrical objects.	PL/SQL query of 3DFDS or STS. Query takes the form: `SELECT TOPOLOGY.QUERY_PAIR(<< FEATURE ID 1>>, << FEATURE ID 2>>) FROM DUAL;`
Create rules to support binary relationship identification between complex objects.	No direct query – however, embedded in the PL/SQL algorithms as the above FEATURE_ID values relate to whole objects and STS has been designed to handle complex objects.

Table 81 – Implementation of Requirements for Binary Topological Relationships

A series of performance tests (Chapter 8) were run against each structure using these queries. Scalability tests, where performance was monitored for increasing numbers of users, determined that for all three structures overall scalability appears linear. Algorithm complexity was then examined, by running queries against increasing dataset sizes ranging from 264 objects to 1.08 million objects. Results were also examined to determine expected performance under a varying workload.

Analysis to identify the most efficient approach to running 3D binary topological queries was carried out (Chapter 9). This involved comparing the results obtained for the each query type across all three structures.

Table 82 below summarises the results obtained for 1.08 million objects and 8 concurrent users.

Test	*Number of Users*	*STS*	*3DFDS*	*Proxy for As-Required*
9-Intersection Pairs	8	0.34134	3.86793	0.386705
Find Intersecting Objects	8	0.17271	2.754576	0.314057
Find Objects with Relationships	8	0.18642	2.879998	0.768027

Table 82 – Summary Test Results, 1.08 million Objects

STS out-performs 3DFDS in all cases. It also out-performs the Proxy for As-Required calculations although the results obtained here are impacted by the selection of the multiplication factor described in Section 8.4.4 and the use of single-user test results for all user combinations. Thus, in the context of the work described in this thesis, STS provides the most efficient structure for binary relationship queries, but further work is required to determine the efficiency of a structure-based approach when compared to an As-Required approach.

The final research question was stated as:

What additional considerations should be made to support the inclusion of 3D topological functionality in 3D GIS?

Product development for this research (Chapter 10) involves the development of the "3D Topology Toolkit". A series of recommendations for further work towards commercialisation were also listed. These included the development of a 3D topology engine and other 3D GIS functionality including visualisation of 3D data and 3D directional and metric queries. The current status of 3D GIS has been reviewed, along with suggested measures to improve its uptake and the importance of identifying early adopters to support the process of product development highlighted.

11.2 Research Differentiation

Researchers including Zlatanova (2000), Molenaar (1990), Rikkers *et al.* (1994), Billen *et al.* (2002) and Pigot (1995) have worked extensively on theory and implementations of topology in a 3D situation. This research differentiates itself from previous work through a number of aspects.

In contrast to previous research, the requirements gathering process took a cross-domain approach rather than being application-specific, resulting in three generic topological queries. A mechanism to relate user-specified terminology for these relationships existing topological frameworks was identified, allowing users to implement their own understanding of topology, which tends to be application domain-specific, within the generic system. This extends

previous approaches, which map relationship names assigned by framework designers to specific framework relationships.

The decision not to develop a 3D topological engine was initially taken to focus the scope of the research and for commercial reasons, but had two additional consequences. Firstly, a Proxy for an As-Required implementation in 3D was developed, combining available functionality in 3D and 2D to provide an indication of expected performance of such an engine. Secondly, the absence of the engine refocused the research towards binary relationship identification, with a secondary focus on data maintenance.

A new structure, STS, was developed to improve query performance for these relationships. This structure differentiates itself from those proposed previously (which are mainly based on the B-Rep approach) due to the reduced number of tables, the reduced number of joins required to identify binary relationships and the reduced storage requirements. Unlike B-Rep, the interest in primitives is whether they are shared between objects, rather than the identification of the next primitive in line to create an ordered list. STS does not enforce a Node/Edge/Face/Volume hierarchy (although this can be imposed by the topological engine if required), allowing greater flexibility when modelling real-world objects. This includes the modelling of non-manifold objects, of curved surfaces and of situations where objects have no spatial representation (identified as one of the requirements for 3D topology to support Emergency Response and Chemistry applications).

The replicated dataset is the first implementation of a comprehensive 3D dataset of binary relationships between simple objects in 3D. Although lacking real-world characteristics such as multi-part geometry, complex geometry or high interconnectivity of objects, it can be used to systematically test approaches to binary relationship identification (structures, algorithms or combinations of these) and ensure that they implement the entire set of 9-Intersection relationships described by (but not implemented by) Zlatanova (2000). As a replicated dataset exists, it can also be used to underpin comparative performance testing of these approaches.

Set-based algorithms were developed for relationship determination using each of the above approaches. These differ from the rules-based approaches described in the literature as the latter are designed to minimise the number of queries performed and may fail to identify valid relationships. The PL/SQL implementation is also the first, to the knowledge of this researcher, that combines and wraps the components of the 9-Intersection matrix in 3D into one R-Value and maps this value to end-user terminology, rather than providing lower level functionality

such as the identification of shared Nodes between objects. This presents a simple interface to the user, rather than requiring specialist knowledge of topology.

Comparative performance tests bring the above concepts together. Again, in contrast to many previous researchers who implement real-world data within the context of a single data structure, these were executed across three structures, and were designed to identify the most efficient approach for the execution of binary topological queries. Testing included both scalability and algorithm complexity tests and a review of the impact of varying query workload was also conducted.

The aim of this research has been to develop a system to integrate within a commercial 3D GIS rather than to solve a specific 3D topological problem or represent a particular data type. The investigation into commercialisation of the work represents the final point of differentiation, presenting an outline business case for further research and development.

11.3 Further Research

A number of areas can be identified as requiring further work following on from the completion of this research.

11.4 Enhancing the 3D Topological Toolkit

The development of a 3D topology engine and the integration of 3D topology with other aspects of 3D GIS including visualisation tools and metric and directional queries are fundamental if 3D topology is to be deployed as part of 3D GIS. Testing the data structure using real-world data also forms part of this process, as does widening the range of relationships supported by the system, extending the range of object types represented in the test dataset, modifying STS to support the data maintenance process and further improve binary query performance.

11.4.1 Developing a 3D Topology Engine

The 3D Topology engine is fundamental to the further development of the 3D Topology toolkit and should be considered a priority in terms of further research. It should be designed to populate and maintain data in STS as rapidly as possible, taking into account the complexity of real-world data. The requirement for modelling relationships between legacy 2D and new 3D data should also be considered - is this best accomplished by temporarily projecting the 3D data into 2D or by extruding 2D data? Such an engine would facilitate tests to compare 9-Intersection relationship performance between a full implementation of the As-Required approach and a structure-based approach. The data structure comparison between 3DFDS and STS can be extended take into account structure population and maintenance times. Although it

may be possible to develop such an engine from scratch, given the complexity of this task other alternatives such as the use of Parasolid's Bodyworks (Parasolid 2007) or the CGAL libraries (CGAL 2007) should also be investigated.

11.4.2 Extending the Test Dataset

As noted in Chapter 2, much of the research described in relation to the 9-Intersection framework has been carried out in the context of 2D datasets. Additional effort is required to determine a complete set of possible 3D 9-Intersection relationships for typical complex and compound objects. This would build on research carried out by Zlatanova (2000) and Schneider and Behr (2006) and also extend the work carried out by Li (2006) who classifies the relationships between multiple closed regions in 2D. As the PL/SQL algorithms implemented are set-based, and thus identify any 9-Intersection relationships, it may be possible to utilise these to evaluate the theoretical relationships and also to identify any relationships existing in real-world data not identified by a constraints-based approach.

The set-based algorithms described in this thesis can also be utilised to validate the results obtained in 2D by authors including Schneider and Behr (2006), Li (2006) and Nguyen *et al.* (1997) for relationships between complex and compound objects, as each component of the relationship will always be evaluated. The data captured for these 2D topological relationship tests can then be added to the replicated dataset if required.

Use of real-world data would allow confirmation that the performance differences observed between the various approaches hold for all datasets. Additionally, the suitability or otherwise of each implementation approach for specific applications could be investigated. Real-world data should be selected to include overlapping, multi-part and geometrically complex objects which relate to many hundreds of primitives and where primitives are shared amongst many objects.

11.4.3 Widening the Range of Relationships

The STS has been tested against the 9-Intersection framework, as this is one of three identified by the Open Geospatial Consortium (OGC) and its use also facilitated mapping between end-user terminology and framework relationships. However, the OGC recommend two additional sets of operators in the context of topology –Full and Set Theoretic. Further investigation is required to ensure that STS supports the implementation of such queries. The Set Theoretic approach utilises the closure and exterior of the object, where the closure is the union of the interior and boundary utilised by 9-Intersection. Thus it is likely that queries using this framework can be built on STS. The Full framework takes into account the highest dimension

of the intersection as well as the interior, boundary and exterior of the objects. It is anticipated that this can be identified by directly querying the TOPO_ tables to identify shared primitives. Extending the implementation described above to take account of the TRUE, FALSE and NULL (do not evaluate) intersection options described by the ISO 19107 may also be required for full compliance.

11.4.4 Enhancing the Simplified Topological Structure

Using the Simplified Topological Structure, information such as which particular Faces a Node or Edge form the boundary of must be derived from the spatial objects representing the primitives. Whilst this issue is not topological in nature, it may impact overall performance for structure population time. Further research, in the context of a working 3D Topology engine, is required to determine whether identifying the relationships between selected primitives using spatial relationship queries results in adequate performance or whether STS should be extended to directly store these relationships.

STS flags any primitive forming the boundary of a part object as a boundary of the whole object. This yields correct results for manifold objects. However, this has limitations in terms of non-manifold objects. Figure 100 exemplifies this issue. Edge E1 is a boundary of Part Object A1 and of Part Object A2. However, it may not represent a boundary primitive of object A as a whole – this will depend on the definition of boundary for the non-manifold object.

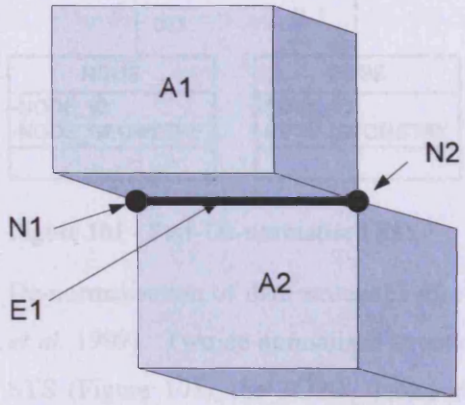


Figure 100 - Determining the Boundary of Non-Manifold Objects

It may be possible to handle these situations by defining separate IS_BOUNDARY flags for the part object and for the object as a whole. This would require modification of the STS, adding another column onto the TOPO_ tables (resulting in columns IS_BOUNDARY_PART, IS_BOUNDARY_WHOLE) and then altering the algorithms to query the appropriate column depending if you are looking at relationship with the whole or the part object.

Both the STS and 3DFDS algorithms start by determining the dimension of the part object, which is required to differentiate between interior and boundary primitives. To improve algorithm performance, it may be also possible to store the dimension of each part object as a field in the TOPO_PART_TABLE, rather than determine this through queries against the TOPO_ tables.

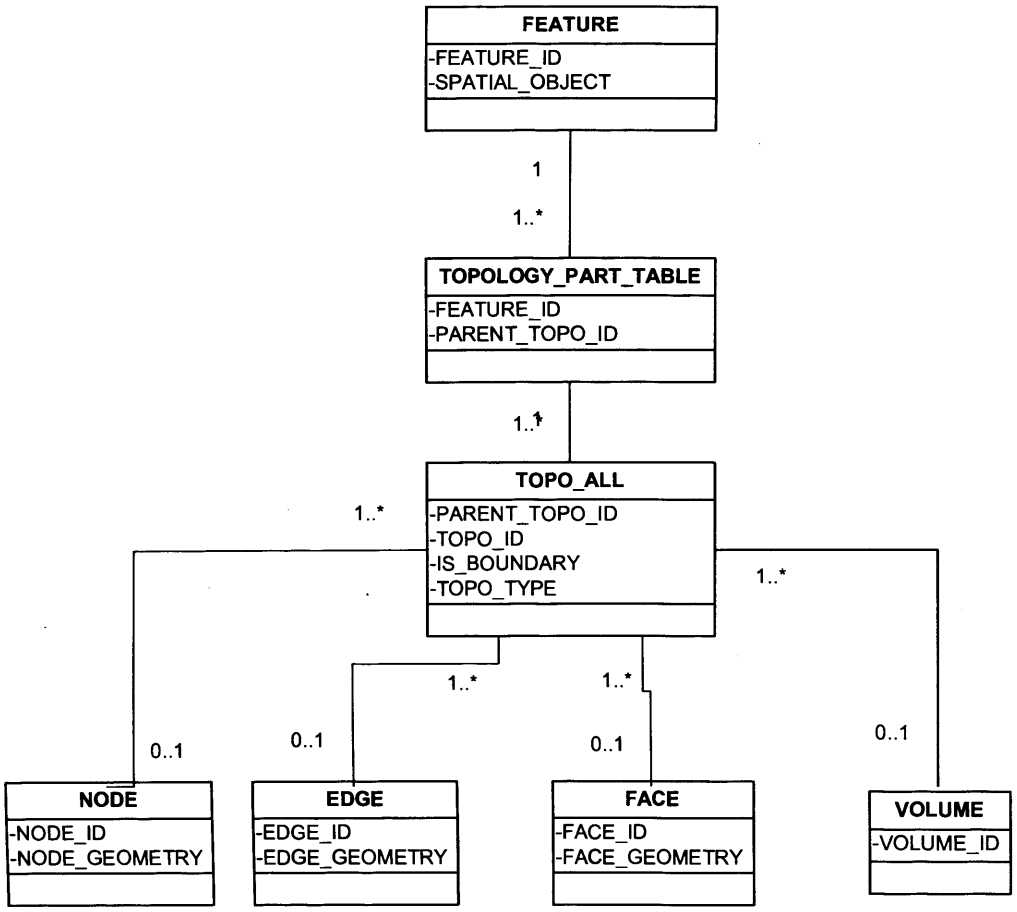


Figure 101 - Part-De-normalised STS

De-normalisation of data structures may improve query performance (Hoxmeier, 1997, Atzeni *et al.* 1999). Two de-normalised structures can be suggested for STS - for Part-De-normalised STS (Figure 101), the TOPO_ tables can be merged into one table, which also contains an additional field to identify the type of topological primitive in question. Binary relationship queries against the structure query the FEATURE, TOPOLOGY_PART_TABLE and TOPO_ALL tables, following a maximum of two joins. For the fully de-normalised structure, the TOPOLOGY_PART_TABLE and TOPO_ALL table are merged. Binary relationship queries now only follow one join.

The existing replicated test dataset can be utilised to populate these structures, and PL/SQL code modified to run tests identical to those described in this thesis. Given that the querying

de-normalised structures further reduces the number of joins to be followed, performance results obtained may be improved when compared to STS as described in this thesis. However, the increased table (and hence index) size may negatively impact performance, and the creation of redundant data (for example, with full de-normalisation the relationship between each FEATURE and PART FEATURE is repeated for every associated primitive) makes the data maintenance process more complex.

11.4.5 Validating and Improving Performance Test Results

As discussed in Chapter 9, predictive analysis using the results obtained from the scalability and algorithm complexity tests was limited both by the number of concurrent user tests and by the availability of only five replicated datasets for each structure. The identification of more than one possible trend line for the results highlights this issue. If accurate predictions are required, further testing using larger volume datasets and higher numbers of concurrent users should be carried out. Anomalous test results also require further investigation, possibly by executing tests in an alternate test environment or re-running the tests. Again, removing these issues would improve the quality of the predictions. An issue with the order of test execution was also identified whereby data was pre-loaded due to the execution of the *9-Intersection Pairs* first. Thus the tests should be re-run with any loaded data being cleared from memory between tests.

The algorithms evaluate all nine components of the 9-Intersection query. Additional investigation is required to compare this approach to a rules or condition based approach, and identify under what circumstances each is optimal – for example, if the data is highly connected or if objects are linked to many topological primitives. It may also be possible to combine the two approaches, evaluating the full set of 9-Intersection components only when a specific R-Value is required but offering a constraints-based approach in other cases. For very large systems, modifying the algorithm such that each component of the 9-Intersection matrix is evaluated in parallel with the others may also be considered.

One advantage of Object-Relational databases not considered as part of this research is the use of Object References, which replace standard referential constraints and have advantages including the simplification of queries. As this approach may improve query performance both 3DFDS and STS should be re-implemented in this manner, and the performance tests re-run.

11.4.6 Using Real-World Data

The tests described in Chapter 7 and the additional work described here should also carried out using real-world data, to ensure that the results are not impacted by the factors described in

Section 9.12.3, including the pair-wise nature of the dataset, planar surfaces and objects containing few Nodes, Edges and Faces.

11.5 Integrating 3D Topology with 3D GIS

Tasks here include further investigation into the requirements for the 3D Topology toolkit, the process of mapping terminology from user-domains to framework relationships and the relevance of 3D topology to other 3D GIS functionality including metric and directional queries, visualisation and generalisation.

11.5.1 Requirements Gathering

To date, the requirements gathering process involved a comprehensive review of literature relating to users of 3D data and 2D topology. This approach was selected due to the general immaturity of commercial 3D GIS - end-users of 3D data do not necessarily understand their work in terms of 3D topology, and thus may not be able to relate this research to their work without a prototype demonstration. Lack of common terminology across 3D application domains is also an issue. Ideally, however, interviews with potential end-users should be carried out to obtain further requirements and query details not available through literature alone. The results may provide further indication as to the uptake prospects of 3D topology, leading to prioritised functionality development.

11.5.2 Mapping End-User Terminology to 9-Intersection Codes

The approach described allows individual end-users to define the topological relationships in which they are interested using application-specific terminology. However, configuring the relationship mapping requires a specialist end-user with knowledge of topological relationships and the 9-Intersection framework. Further research is required into methods to simplify this process to allow non-specialist users to create their own task-specific mapping. This may involve the development of a user interface to present pictorial examples of the relationships, allowing the user to select a relationship diagram and associate this with a specific description.

11.5.3 Developing Metric Queries

Metric queries require the development of distance, area and volume calculation algorithms using a similar approach to that taken for the binary topological relationships in order to allow them to be incorporated into SQL statements. Such algorithms can take advantage of the database's in-built spatial object types to represent whole objects in order to avoid reconstructing the object from associated topological primitives.

11.5.4 Support for Routing Algorithms

STS has been designed to optimise performance for binary relationship queries as these underpin queries of higher order. STS also supports simplex primitives and can thus be utilised to implement triangle and tetrahedron-based structures. A requirement to model relationships beyond two Features was identified as part of the review process, in particular to support 3D pedestrian routing for emergency planning. The possibility of extending STS to support network algorithms such as shortest path (adding direction to the Edge primitives) can be considered. However, it may be more appropriate to implement a separate data model optimised to support these types of queries, deriving the information to populate the model from 3D binary relationship queries carried out against STS.

As STS does not derive boundary information from links between primitives but encodes this information in the structure, it may be possible to flag primitives to assist navigation through built environments, adding a third option to the IS_BOUNDARY flag. Although both walls and doors form the boundary of a room, if the door is flagged as IS_BOUNDARY = CONNECTED this information can be used to directly identify the navigation points between rooms. Any remaining shared walls can be flagged as IS_BOUNDARY = TRUE, differentiating between navigable connections and shared (adjacent) walls. For R-Value determination, the CONNECTED value would have the same meaning as the TRUE value.

This level of connectivity information cannot be encoded in a B-Rep structure, as all walls are automatically flagged as boundary primitives, having dimension one less than the 3D room. Connectivity is currently be derived through a more extended query, first identifying the shared primitives and then selecting the subset of these flagged as 'door' objects.

11.5.5 Developing Directional Queries

Directional queries again require knowledge of the object geometry. Further work is required to translate this requirement into terms that can be implemented within the context of a 3D GIS query. This would extend into 3D the work carried out in 2D by Mark and Egenhofer (1994), Sharma (1996), Li (2006b) and Papadias and Theodoridis (1994) amongst others. As with metric queries, to allow full integration, these queries must be developed in such a manner to allow them to be incorporated into a SQL query.

11.5.6 Developing Time-Based Queries

Time-based queries and time sequences form an important of GIS. Simplistically, time-based GIS flag FEATUREs with a 'birth' and 'death' date and time to denote their creation and destruction. Investigation is required to determine whether this concept can be extended to 3D

topological primitives, with primitives being born as soon as a parent object is created, and dying when all associated parent objects have been marked as destroyed.

11.5.7 Visualisation of 3D Data

Due to some similarity between the primitives used in STS and those underpinning 3D visualisation (Nodes, Edges and Faces), it may be possible to utilise topological primitives for visualisation purposes. 3D visualisation algorithms utilise triangular facets, which are supported by STS although their use would result in larger data storage requirements. Alternatively, it may be more efficient to apply triangulation algorithms to the STS primitives rather than the parent object geometry – STS primitives are simpler in structure and as they are shared between objects the total number of coordinates to process is also reduced. Using primitives for visualisation may remove the requirement to store the whole object SDO_GEOMETRY alongside the primitives, reducing data duplication and storage requirements.

11.5.8 Generalisation

Displaying different levels of detail at different scales is an important concept in 3D visualisation and is implemented within 2D GIS using functionality to switch data layers on and off at specific display scales. In 3D, a similar concept could also be applied. Further work is required to determine the possibility of generalising datasets to support this functionality, making use of the topological data structure to identify and maintain topological relationships at the different display scales (for example, assuming a building is always north of a river) and also to simplify 3D geometry by removing internal holes and cavities.

11.6 Beyond GIS

To conclude the review of areas of further work arising from this thesis, two non-GIS applications can also be considered.

11.6.1 Combining Concepts – 3D Gaming and Virtual Reality

The concept of utilising topological primitives to support 3D visualisation can be taken further to include 3D Gaming and Virtual Reality systems. Traditional 3D gaming makes use of proprietary systems for data storage, to provide enhanced rendering performance. Additionally, games tend to be graphics intensive and do not require depicted objects to have much non-graphic attribution. Development focuses on rapid data retrieval and display as well as realism and end-user interaction. Further research is required to determine if it is possible to combine these concepts with the primitives provided by a topological structure within a 3D GIS. For example, could existing 3D R-Tree filtering be utilised to select appropriate primitives for

visualisation in a particular scene? Could the required levels of detail be provided by topology-enhanced generalisation algorithms? What value would be added to games if a rich set of attributes were associated with each displayed object? Would the performance obtained be sufficient?

11.6.2 Extending STS to the Fourth Spatial Dimension

The fourth spatial dimension should be distinguished from what is commonly called the fourth dimension – i.e. time. The simplicity of STS lends itself to easy extension into the former, as shown in Figure 102 below. The 4D primitive in this case could be known as a choron.

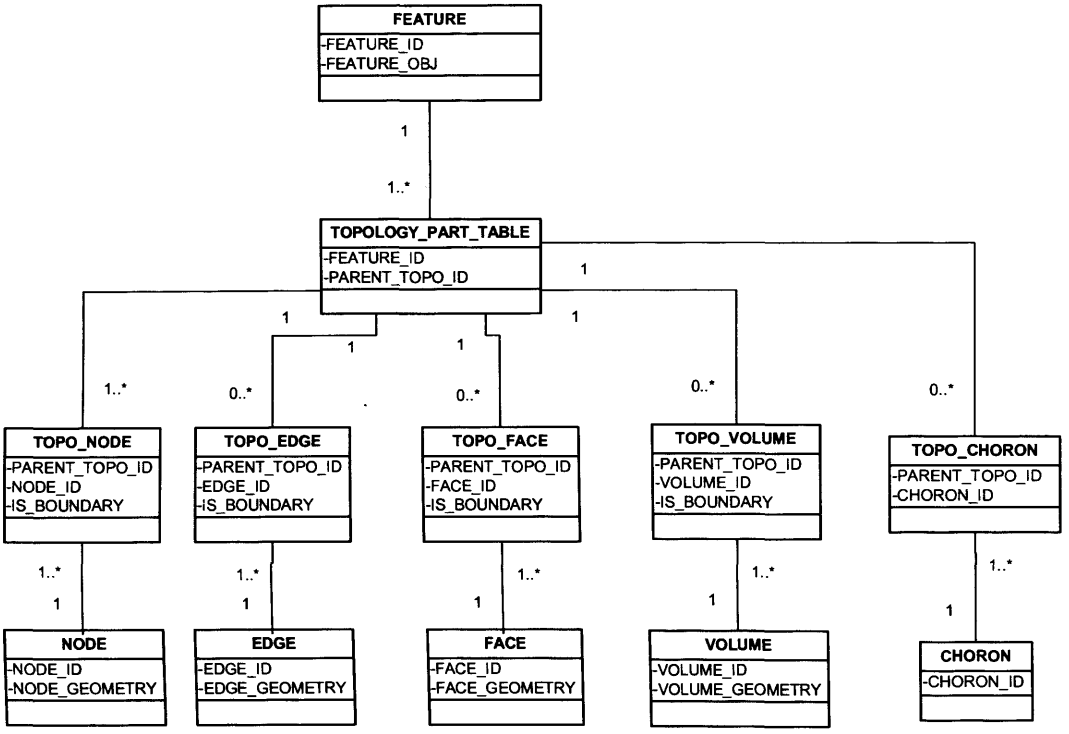


Figure 102 - STS extended to the fourth spatial dimension

Such an extension would allow the examination of binary topological relationships between 4D objects (known as polychora or 4-polytope, described in Banchoff, 1990). Extending the existing approach, two polychora would be adjacent when they share the same Volume primitive, and the IS_BOUNDARY flag on the Volume primitive would denote whether the primitive formed part of the boundary of the 4D object or was contained within the object.. The possibility of applying frameworks such as 9-Intersection to such objects could also be examined. As it is currently not possible to represent 4D objects within a relational database environments, such work could utilise STS without spatially representing the polychora. Extending this concept further, investigation would also be required into how to represent these objects in the database and create the corresponding R-Trees required for indexing. To drive the research, applications utilising four spatial dimensions should be identified.

11.6.3 Integrating Space and Time using STS

As described in Section 5.7, it is not necessary to represent primitives in STS spatially. Van Oosterom *et al.* (2006) identify two time-related primitives – point-in-time and time-interval. These could be represented in STS as two non-spatial primitive types, where point-in-time forms the boundary of time-interval. Figure 103 gives the resulting version of STS.

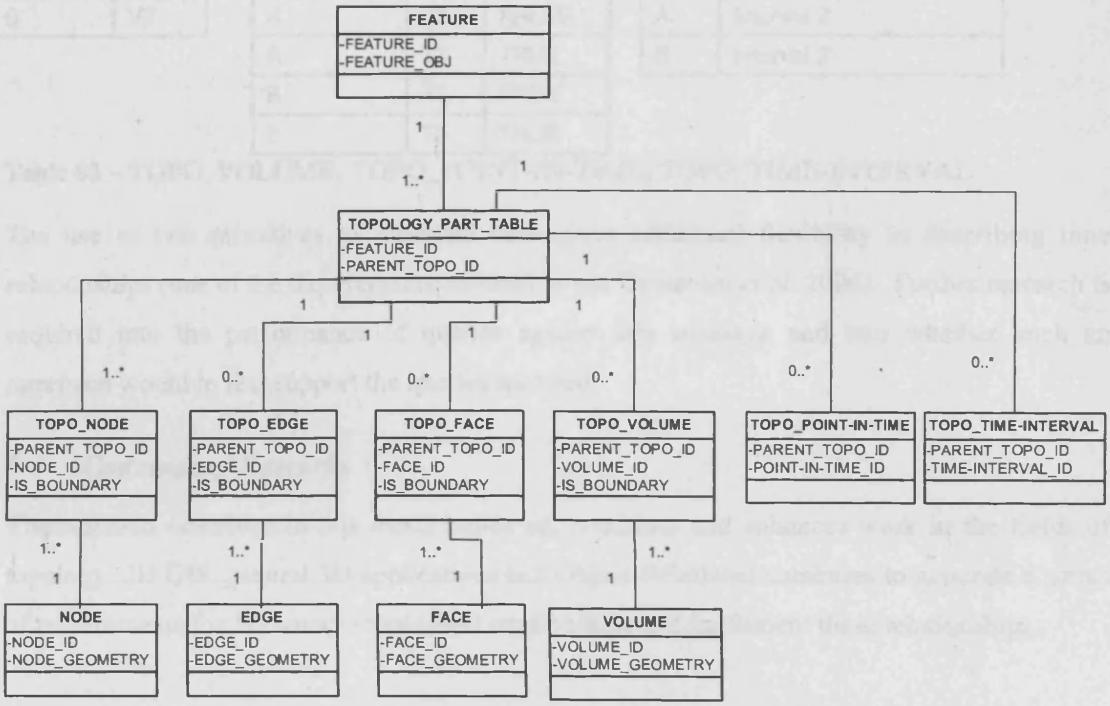


Figure 103 - Integrating STS and Time Primitives

An example is given here to demonstrate how time primitives could be integrated with spatial objects for query purposes, for the situation in Figure 104, where building A is built at T1, extension B is added at T2 and both are demolished at T3 (to simplify the description, only the Volume primitives are shown here).

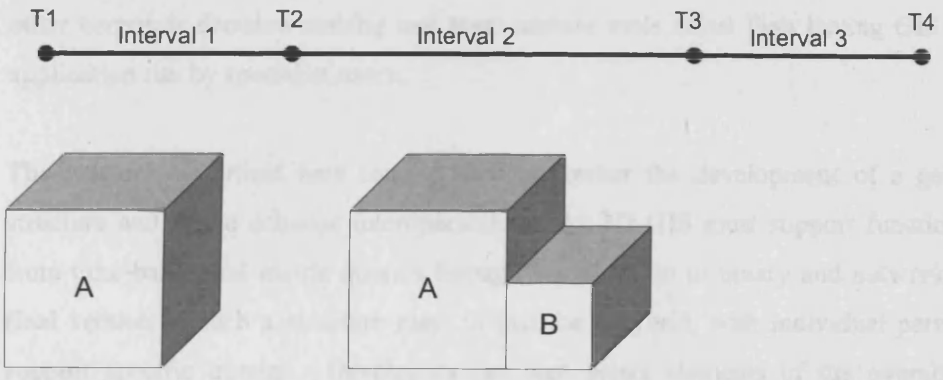


Figure 104 – STS and Time – an Example

TOPO_PART_ID	VOLUME_ID
A	V1
B	V2

TOPO_PART_ID	POINT-IN-TIME_ID	IS_BOUNDARY
A	T1	TRUE
A	T2	FALSE
A	T3	TRUE
B	T2	TRUE
B	T3	TRUE

TOPO_PART_ID	TIME_INTERVAL_ID
A	Interval 1
A	Interval 2
B	Interval 2

Table 83 – TOPO_VOLUME, TOPO_POINT-IN-TIME, TOPO_TIME-INTERVAL

The use of two primitives to represent time gives additional flexibility in describing time relationships (one of the requirements outlined in van Oosterom *et al.* 2006). Further research is required into the performance of queries against this structure and into whether such an approach would in fact support the queries required.

11.7 Concluding Remarks

The research described in this thesis builds on, combines and enhances work in the fields of topology, 3D GIS, general 3D applications and Object-Relational databases to generate a series of requirements for 3D binary topological relationships and implement these relationships.

STS and the associated algorithms provide an efficient mechanism for the determination of binary topological relationships, and can be implemented within any Object-Relational database. The diverse number of applications utilising topology - some of which currently employ proprietary data structures and storage mechanisms - highlights the need for standardisation of data structure and functionality. The proposed database-centric architecture results in a system that can take advantage of the trend for integrating GIS functionality in with other corporate decision-making and asset capture tools rather than having GIS as an isolated application run by specialist users.

The research described here can be used to further the development of a generic 3D data structure and hence enhance interoperability. As 3D GIS must support functionality ranging from time-based and metric queries through visualisation to binary and network topology, the final version of such a structure may, in fact, be a hybrid, with individual parts optimised to support specific queries. Developers can then select elements of the overall structure for implementation as required. Although the hybrid approach involves additional data maintenance issues (as the redundant storage of coordinate information may yield to

inconsistent data) it may prove more suitable in terms of performance than a one-size-fits-all approach.

The work described here may benefit researchers in the field of 3D GIS, to the developers of commercial 3D GIS and end-users of such systems, be they based in the Cadastral, Earth Science, Architectural or other domains – in fact, it is unlikely that the development of a commercial 3D GIS will be underpinned by a single application area. It is hoped that one of the key outcomes of this research is the ability to familiarise users with the functionality that could be offered by topology in 3D, thus breaking away from the current "lack of requirements impedes functionality development which in turn impedes requirements identification" situation.

References

ABDELMOTI A, EL-GERESY B, 1995, A general approach to the representation of spatial relationships, in *Proceedings of the International Conference on Information and Knowledge Management CIKM 95*, ACM Press

ABDELMOTY, A, WILLIAMS M, 1994, Approaches to the Representation of Qualitative Spatial Relationships for Geographic Databases: A Critical Survey and Possible Extensions, in MOLENAAR, M, DE HOOP, S, eds., *Advanced Geographic Data Modeling*, Netherlands Geodetic Commission, Publications on Geodesy No. 40

AFSHAR, M, BOROUMAND, M, STUDNICKA, N, 2002, Archaeological Scanning of Persepolis, *GIM International* 16(6), 12–14

AGARWAL, P, 2005, Topological operators for ontological distinctions: disambiguating the geographic concepts of place, region and neighbourhood, *Spatial Cognition and Computation*, 5 (1), 69-88

AHO, A, HOPCROFT J, ULLMAN, J, 1987, *Data Structures and Algorithms*, Addison Wesley Publishing, London

ALLAMARAJU S, LONGSHAW A, O'CONNOR D, VAN HUIZEN G, DIAMOND J, GRIFFIN J, HOLDEN M, DALEY M, WILCOX M, BROWETT R, 2000, *Professional Java Server Programming – J2EE Edition*, Wrox Publishing

APEL, M, 2001, Development of a 3D GIS based on the 3D Modeller Gocad, in *Proceedings of the International Association for Mathematical Geology Annual Meeting*, Cancun, Mexico 256-268

APEL, M, 2006, From 3D Geomodelling Systems Towards 3D Geoscience Information Systems: Data Model, Query Functionality, and Data Management, *Computers and Geosciences*, 32, 222-229

ARENS, C, 2003, *Maintaining Reality – Modelling 3D Spatial Objects in a Geo-DBMS Using a 3D Primitive*, Masters Thesis, Delft University of Technology

ARENS, C, STOTER, J, Van OOSTEROM, P, 2005, Modelling 3D Spatial Objects in a geo-DBMS using 3D Primitive, *Computers & Geosciences*, 31, 165-177

ATZENI, P, CERI, S, PARABOSCHI, S, TORLONE, R, 1999, *Database Systems – Concepts, Languages and Architectures*, London, The McGraw Hill Companies

AUTODESK, 2006, *Autodesk Map3D*, [online], Available from http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=3081357, [Accessed 20th October 2006]

AVS 2006, ADVANCED VISUAL SYSTEMS, [online], Available from http://www.avs.com, Accessed 20th October 2006

BAARS, M, STOTER, J, van OOSTEROM, P, VERBREE, E, 2004, Rule - based or explicit storage of topology structure : a comparison case study, in TOPPEN, F, PRASTACOS, P, eds., Proceedings of The 7th Conference on Geographic Information Science, Crete University Press, 765-769

BANCHOFF, T, 1990, *Beyond the Third Dimension*, Scientific American Library, New York

BARCELO, A, DE CASTRO, O, TRAVET, T, VICENTE, O, 2003, A 3D Model of an Archaeological Excavation. In DOERR, M, SARRIS, A, eds. *The Digital Heritage of Archaeology, Computer Applications and Quantitative methods in Archaeology*, Hellenic Ministry of Culture, Archive of Monuments and Publications

BELLOSO, L, CHARCARTEGUI, F, CORTIULA, B, ESCORCIA, F, MATA, T, SAMPSON, E, CASCO, R, HUSBAND, J, MONSEGUI, G, TAYLOR, C, LESSO, B, SUAREZ, T, 1994, *Teamwork Renews and Old Field with a Horizontal Well*, [online]. Available from:, http://www.oilfield.slb.com/media/resources/oilfieldreview/ors94/0494/p59_69.pdf, [accessed 12 December 2004]

BENHAMU, M, DOYTSHER, Y, 2003, Towards a Spatial 3D Cadastre in Israel, *Computers, Environment and Urban Systems*, 27 (4), 359-374

BILLEN, R, ZLATANOVA, S, 2003, Conceptual Issues in 3D Urban GIS, *GIM International*, 17(1), 33-35

BILLEN, R, ZLATANOVA, S, MATHONET, P, BONIVER, F, 2002, The Dimensional Model: a framework to distinguish spatial relationships, in *Proceedings of ISPRS Commission IV Symposium on Geospatial Theory, Processing and Applications*, Ottowa, Canada

BORRMANN, A, VAN TREECK, C, RANK, E, 2006, Towards a 3D Spatial Query Language for Building Information Models in Proc. of the 11th Int. Conf. on Computing in Civil and Building Engineering (ICCCBE)

BREUNIG, M, ZLATANOVA, S, 2005, 3D Geo-DMBS, in ZLATANOVA, S, PROSPERI, D, eds., *Large-scale 3D Data Integration*, Taylor and Francis, London

BRISSON, E, 1989, Representing Geometric Structures in d Dimensions: Topology and Order, in *Proc. Fifth Annual Symposium on Computational Geometry*, Saarbruchen, West Germany, 218-227

BROWN, D, 2002, Topology and Chemistry, *Structural Chemistry* 13 (3-4), 339-355

BROWN, R, 1988, *Topology – A Geometric Account of General Topology, Homotopy Types and the Fundamental Groupoid*, Ellis Horwood Ltd, Chichester, England

CARLSON E, 1987, Three Dimensional Conceptual Modelling of Subsurface Structures, *Technical Papers ACSM-ASPRS Annual Convention*, 4, 188 – 200

CATHEY R J, BEITZEL S M, JENSEN E C, PILOTTO A J, GROSSMAN D, 2004, Measuring the Scalability of a XML-QL Relational Database Management System, in: *Proceedings of the 2004 IEEE International Conference on Information Technology - Coding and Computing (ITCC)*, Las Vegas

CGAL, 2007, *Computational Geometry Algorithms Library*, [online], Available from: http://www.cgal.org, [Accessed 23rd September 2007]

CHAZELLE, B, DOBKIN, D, 1987, Intersection of Convex Objects in Two and Three Dimensions, *Journal of the Association of Computing Machinery (ACM)*, 34,1, 1-27

CHEN, J, LI, Z, LI, C, GOLD, C, 2001, Describing Topological Relations with Voronoi-Based 9-Intersection Model, in FRITSCH, M, ENGLICH, M, SESTER, M, eds., *International Archives of Photogrammetry and Remote Sensing*, 32(4), 99-104

CHRISMAN, N, DOUGENIK, J, WHITE, D, 1992, Lessons for the Design of Polygon Overlay Processing from the ODYSSEY WHIRLPOOL Algorithm, in *Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, South Carolina*, 401-410 International Geographical Union IGU

CITYGML, 2006, City GML (City Geography Markup Language), Draft Specification, Version 0.3.0, edited by GROGER, G, KOLBE, T, CZERWINSKI, 120 pages, [online], Available from http://www.citygml.org/docs/CityGML_Specification_0.3.0_OGC_06-057.pdf [accessed 24th August 2006]

CLEMENTINI, E, DI FELICE, P, CALIFANO G, 1995 Composite Regions in Topological Queries, *Information Systems*, 20(7), 579-594

CLEMENTINI, E, DI FELICE, P, VAN OOSTEROM, P, 1993, A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In ABEL, D, OOI, B, eds., *Advances in Spatial Databases*, Springer Verlag, 277-295.

CLEMENTINI, E, SHARMA J, EGENHOFER M, 1994, Modeling Topological Spatial Relations: Strategies for Query Processing, *Computers and Graphics*, 18(6), 815-822

COHN, A, BENNET, B, GOODAY, J, GOTTS, N, 1997, Qualitative Spatial Representation and Reasoning with the Region Connection Calculus, *Geoinformatica*, 1,1-44

COORS, V, 2003, 3D-GIS in Networking Environments, *Computers Environment and Urban Systems*, 27, 345-357

CORBETT, J, 1979, Topological Principles in Cartography. *Technical Paper No. 48*, U.S. Bureau of the Census. cited in GALDI, D, 2005, Spatial Data Storage and the Topology in the Redesigned MAF/TIGER System , US Census Bureau, [online], Available from http://www.census.gov/geo/mtep_obj2/topo_and_data_stor.html, [Accessed 16th April 2006]

DARATECH, 2002, GIS Software Growth Hits A Dynamic 14.3%, *Daratech Press Release*, [online], Available from: http://www.directionsmag.com/press.releases/index.php?duty=Show&id=5946, [Accessed 17th March 2007]

DARATECH, 2004, *Daratech Forecasts Worldwide GIS Revenue to Top $2.02 billion in 2004, up 9.7% over 2003,* [online], Available from: http://www.directionsmag.com/press.releases/index.php?duty=Show&id=10412&trv=1, [Accessed 20th October 2006]

DARATECH, 2005, Lack of Standards, Web-Proliferation, Homeland Security, Information Management Needs Changing the Face of GIS and Geospatial Industries), *Daratech Press Release,* [online], Available from: http://www.daratech.com/press/releases/2005/050328.html, [Accessed 20th October 2006]

DARATECH, 2006a, *Extract from GIS/Geospatial Markets & Opportunities,* [online], Available from http://www.daratech.com/presss/releases/2006/060828.html, [Accessed 20th October 2006]

DARATECH, 2006b, GIS/Geospatial Public Sector Software Market forecast to surpass the regulated sector in 2006, Daratech Press Release, [online], Available from *http://www.directionsmag.com/press.releases/index.php?duty=Show&id=15043&trv=1,* [Accessed 24th October 2006]

DARATECH, 2006c, GIS/Geospatial Market Grew 17% In 2005 To Top $3.3 Billion; Sales Led By Growth In Data Products; *Daratech Press Release,* [online[, Available from http://www.directionsmag.com/press.releases/index.php?duty=Show&id=14697, [Accessed 12th March 2007]

DATE, C, 1990, *An Introduction to Database Systems*, Wokingham, England Addison-Wesley

DE FLORIANI, L, 2003, Solid Modeling: Part two (boundary schemes), [online], Available from http://www.cs.umd.edu/class/fall2003/cmsc828D/solidmodeling-828D-2.pdf, [Accessed 7th December 2004]

DE HOOP, S, VAN OOSTEROM, P, MOLENAAR, M, 1993, Topological Querying of Multiple Map Layers, *Proceedings of COSIT, European Conference on Spatial Information Theory*, 139-157

DE LA LOSA, A, CERVELLE, B, 1999, 3D Topological Modelling and Visualisation for 3D GIS, *Computers & Graphics,* 23, 469-478

DEMERS, M, 1997 *Fundamentals of Geographic Information Systems*, John Wiley and Sons, Inc. New York

DUNBAR, M, 2003, 3D Visualization for the Analysis of Forest Cover Change, in SCHIEWE J, HAHN M, MADDEN, M, SESTER, M, eds., *Proceedings ISPRS Commission IV Joint Workshop: Challenges in Geospatial Analysis, Integration and Visualization II*, Stuttgart, Germany

EGENHOFER M, 1989, A Formal Definition of Binary Topological Relationships, In LITWIN, W, SCHEK, H, (eds.), *Third International Conference on Foundations of Data Organisations and Algorithms,* Lecture Notes in Computer Science, 367, 457-472

EGENHOFER, M, CLEMENTINI, E, DI FELICE, P, 1994, Topological Relations between Regions with Holes, *International Journal of Geographical Information Systems,* 8(2), 129-142

EGENHOFER, M, FRANZOSA, R, 1991, Point-Set Topological Spatial Relations, *International Journal for Geographical Information Systems*, 5(2), 161-174.

EGENHOFER, M, FRANZOSA, R, 1994, On the Equivalence of Topological Relations, *International Journal of Geographical Information Systems,* 8(6), 133-152

EGENHOFER, M, GLASGOW, J, GUNTHER, O, HERRING, J, PEUQUET,D, 1999, Progress In Computational Methods For Representing Geographic Concepts, *International Journal of Geographical Information Science*, 13(8), 775-796

EGENHOFER, M, HERRING, J, 1990, Categorizing Binary Topological Relations between Regions, Lines and Points in Geographical Databases, *NCGIA Technical Report*

EGENHOFER, M, SHARMA J, MARK D, 1993, A Critical Comparison of the 4-Intersection and 9-Intersection Models for Spatial Relations: Formal Analysis, in MC MASTER, R, ARMSTRONG, M, eds., *Proceedings of Autocarto 11,* Minneapolis

EL-HAKIM, S, BERALDIN, J, PICARD, M, 2003, 3D Modeling of Heritage Monuments. *GIM International* 17(4), 13–15.

ELKADI, A, HUISMAN, M, 2002, 3D-GSIS, Geotechnical Modelling of Tunnel Intersection in Soft Ground: the second Hienenoord Tunnel, Netherlands, *Tunnelling and Underground Space Technology* 17, 363-369

ELLUL, C, HAKLAY, M, 2006, Requirements for Topology in 3D GIS, *Transactions in GIS*, 10(2)

ELROI, D, 1998, Integration of 2D and 3D Data between GIS and Mine Planning Software, presented at: *Society of Mining, Metallurgy, and Engineering National Convention*, Orlando, Florida, [online], Available from: http://www.elroi.com/pub_pdf.html [Accessed 12 December 2004]

EL-SANA, J, VARSHNEY, A, 1998, Topology Simplification for Polygonal Virtual Environments, *IEEE Transactions on Visualization and Computer Graphics*, 4(2), 133-144

ESRI, 2006, *ArcGIS – The Complete Geographic Information System*, [online], Available from http://www.esri.com/software/arcgis/index.html, [Accessed 20th October 2006]

ESRI, 2006, Working with Geodatabase Topology [online]. Available from http://www.esri.com/library/whitepapers/pdfs/geodatabase-topology.pdf [Accessed 14 June 2006]

FRANK, A, KUHN, W, 1986, Cell Graphs: A Provable Correct Method for the Storage of Geometry, In MARBLE, D, ed., *Proceedings of Second International Symposium on Spatial Data Handling*, Seattle, Washington, 411 – 436

FROST, SULLIVAN, 1998, *European Geographic Information Systems Markets, Report Number 3535-70*, Frost and Sullivan, London

GAIANI, M, GAMBERINI, E, TONELLI, G, 2002, A framework to use virtual worlds generated from real world 3D models as Work Tool for Architectural & Archaeological Restoration on the Web, in *International Journal of Design Computing* (4), [online], Available from: http://www.arch.usyd.edu.au/kcdc/journal/vol4/index.html, [Accessed 12 December 2004]

GALDI, D, 2005, Spatial Data Storage and the Topology in the Redesigned MAF/TIGER System , US Census Bureau, [online], Available from http://www.census.gov/geo/mtep_obj2/topo_and_data_stor.html, [Accessed 16th April 2006]

GE ENERGY, 2006, *Smallworld 4 Product Suite*, [online], Available from http://www.gepower.com/prod_serv/products/gis_software/en/smallworld4.htm, [Accessed 20th October 2006]

GOLDSTEIN D, 2003, Growth Strategies for the Corporate GIS Market *Directions Magazine*, [online], Available from: http://www.directionsmag.com/article.php?article_id=351, [Accessed 17th March 2007]

GONG, J, CHENG, P, WANG, Y, 2004, Three-dimensional modelling and application in geological exploration engineering, *Computers and Geosciences*, 30(4), 391-404

GOODCHILD, M, 1990, *Geographical Data Modeling*, NCGIA Technical Paper 90-11

GOOGLE, 2007, *What is Google SketchUp 6?*, [online], Available from http://sketchup.google.com/product_suf.html, [Accessed 13th March 2007]

GORAWSKI, M, CHECHELSKI, R, 2005, Online Balancing of a R-Tree Indexed Distributed Spatial Data Warehouse, in WYRZYKOWSKI, R, DONGARRA, J, MEYER, N, WASNEIWSKI, J, eds., *Parallel Processing and Applied Mathematics*, Lecture Notes in Computer Science 3911

GOTTS, N, GOODAY, J, COHN, C, 1996, A connection based approach to commonsense topological description and reasoning, *The Monist: An International Quarterly Journal of General Philosophical Inquiry Topology for Philosophers*, 79 (1), 51-75

GRIGNI, M, PAPADIAS, D, PAPADIMITRIOU CH, 1995, Topological Inference, in *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, Morgan Kaufmann, 901-906

GRINSTEIN, R, 2003, A Real World Experiment in 3D Cadastre, *GIM International*, 17(9), 65-67

GROSS, M, 1998, Computer Graphics in Medicine: From Visualization to Surgery Simulation, *ACM SIGGRAPH Computer Graphics*, 32(1), 53-56

GRUEN, A, WANG, X, 1998, CC-Modeler: a topology generator for 3-D city models, *ISPRS Journal of Photogrammetry & Remote Sensing*, 53, 286-295

GRUNWALD, S, BARAK, P, 2003, 3D Geographic Reconstruction and Visualization Techniques Applied to Land Resource Management, *Transactions in GIS*, 7(2), 231-241

GUTING R,1988, Geo-Relational Algebra : A Model And Query Language For Geometric Database Systems, in SCHMIDT, J, CERI, S, MISSIKOFF, N, eds., *Advances in Database Technology: Proceedings of International Conference on Extending Database Technology*, 506-527

GUTTMAN A, 1984, R-Tree: A Dynamic Index Structure for Spatial Search, in *Proc. ACM SIGMOD*, 47-57

HAARSLEV, V, MOLLER R, 1997, SBox – A Qualitative Spatial Reasoner – Progress Report, in IRONI, L, ed., *Proceedings of the 11th IEEE Symposium on Qualitative Reasoning*, Cortona, Italy

HASEGAWA, S, SATO, M, 2004, Real-time Rigid Body Simulation for Haptic Interactions Based on Contact Volume Polygonal Objects, in CANI, M, SLATER, M, (*eds.*) *Eurographics,* 23,3, 530-535

HENDERSON, R, CLARK, K, 1990 Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms, *Administrative Science Quarterly*, 35, 9-30

HERTEL, S, MANTYLA, M, MEHLHORN, K, NIEVERGELD, J, 1984, Space Sweep Solves Intersection of Convex Polyhedra, *Acta Informatica*, 21,5, 501-519

HOEL, E, MENON, S, MOREHOUSE, S, 2003, Building a Robust Relational Implementation of Topology [online]. Available from http://downloads.esri.com/support/whitepapers/ao_/BuildingRobustTopologies.pdf [Accessed 12 December 2004]

HOFFMAN, C, 1989, *Geometric and Solid Modelling: an introduction*, Morgan Kaufman Publications

HOU, E, WU, L, 2002, An Object-Oriented Vector Data Model for 3D Geological Volume Modelling, in *Proceedings of ISPRS Commission II, Symposium 2002, Xi'an P.R. China* August 20-23 2002

HOXMEIER J, 1997, A framework for assessing database quality, *16TH International Conference on Conceptual Modeling (ER97), Workshop 4*, [online], Available from: http://osm7.cs.byu.edu/ER97/workshop4/jh.html, [Accessed 28th August 2006]

HUSEBY, O, THOVERT, J, ADLER, P, 1997, Geometry and Topology of Fracture Systems, *Journal of Physics* 30, 1415-1444

IDL, 2006, *IDL Data Analysis and Visualization Platform*, [online], Available from http://www.ittvis.com/idl/, accessed 20th October 2006

INFORMIX, 2007, IBM Informix Spatial Datablade Module [online], Available from: http://publib.boulder.ibm.com/epubs/pdf/9119.pdf, [Accessed 14th March 2007]

INTERGRAPH, 2006, *Geomedia Professional,* [online], Available from http://www.intergraph.com/gmpro/default.asp, [Accessed 20th October 2006]

JACOBSON, J, VADNAL, J, 1999, Multimedia in Three Dimensions for Archaeology, Information Retrieval with Interactive Models, in *Proceedings of the Systemics, Cybernetics and Informatics Conference and the Conference of Information Systems, Analysis and Synthesis, 1 August – 3 August 1999,* [online], Available from: http://www.planetjeff.net/IndexDownloads/sci-isas-99.pdf, [Accessed 12 December 2004]

JANOSCH, U, COORS, V, KRETSCHMER, U, 2001, Virtual Tourist Guide: Advanced Applications of 3D GIS Orientated towards a Broad Consumer Market. *GIM International*, 15(6), 39-41

KAINZ, W, EGENHOFER, M, GREASLEY I, 1993, Modeling Spatial relations and Operations with Partially Ordered Sets, *International Journal of Geographical Information Systems*, 7 (3), 215-229

KALRA, P, BEYLOT, P, GINGINS, P, MAGNENAT-THALMAN, N, VOLINO P, HOFFMEYER, P, FASEL, P, TERRIER, F, 1995, Topological Modeling of Human Anatomy Using Medical Data, in *Proceedings IEEE Computer Animation Conference* Geneva, Switzerland, [online], Available from: http://www.miralab.unige.ch/papers/63.pdf, [Accessed 12 December 2004]

KOFLER M, GERVAUTZ M, GRUBER M, 2000, R-trees for organizing and visualizing 3D GIS databases. *Journal of Visualization and Computer Animation* 11(3),129-143

KOLBE, T, 2005, Interoperable integration of 3D models over the internet for emergency preparedness and response, *Directions Magazine,* [online], Available from: http://www.directionsmag.com/article.php?article_id=2038, [Accessed 17th March 2007]

KONIDARIS, G, MEHLHORN, K, SHELL, D, 2004, An Optimal Algorithm fr Finding Edge Segment Adjacencies in Configurations of Convex Polygons, [online], Available from: http://domino.mpi-inf.mpg.de/intranet/ag1/ag1publ.nsf/ListPublications?OpenAgent&author=Mehlhorn,+Kurt, [Accessed 2nd April 2007]

KONINGER, A, BARTEL, S, 1998, 3D GIS for Urban Purposes, *GeoInformatica.* 2 (1), 79-103

KRAMER, A, 2002, Topological Coding and Visualization Grammar of the Development of C. elegans, in *The 6th World Multi-Conference on Systemics, Cybernetics and Informatics*, Orlando, Florida, [online], Available from: http://busselab.uni-kiel.de/Site1/downloads/selectedpublication/akraemerPaperID324EC_SCI2002.pdf, [Accessed 12 December 2004]

KRIVOGRAD, S, ZALIK, B 2000, A determination of topological relationships in GIS applications, in *Proceedings of the 22nd International Conference on Information Technology Interfaces*, 377-382

KUFONIYI, O., 1995, *Spatial coincidence modelling, automated database updating and data consistency in vector GIS*, PhD thesis, ITC, The Netherlands, cited in Zlatanova, S, (2000), *3D GIS for Urban Development*, ITC Dissertation Series No. 69

KURATA,Y, EGENHOFER, M, 2006, Topological Relations of Arrow Symbols in Complex Diagrams, in BARKER PLUMMER, D, COX, R, SWOBODA, N, eds., *Diagrams 2006 - Fourth International Conference on the theory and application of diagrams*, Lecture Notes in Computer Science, 4045, 112-126

KWAN, M, LEE, J, 2005, Emergency Response after 9/11: The potential of real-time 3D GIS for quick emergency response in micro-spatial environments, *Computers, Environment and Urban Systems*, 29, 93-113

LADNER, R, ABDELGUERFI, M, WILSON, R, BRECKENRIDGE, J, MCCREEDY, F, SHAW, K, 2001, A Framework for Databasing 3D Synthetic Environment Data. In HC MAYR *et al.*, eds., *Database and Expert Systems Applications* 2113,432-411

LAPIERRE, A, 2007, Response to Geospatial Industry Survey, *GIS Monitor*, [online]. Available from http://www.gismonitor.com/news/newsletter/archive/archives.php?issue=20070125&style=web&length=full#gissurvey, [Accessed 13th February 2007]

LASER-SCAN, 2007, Radius Topology, [online], Available from http://www.1spatial.com/products/radius_topology/?PHPSESSID=ec8623b7c6143f610bd88121fcf8194f [Accessed 15th March 2006]

LAURINI, R, 1998, Spatial Multi-Database Topological Continuity and indexing: a step towards seamless GIS data interoperability, *International Journal of Geographical Information Science*, 12 (4), 373- 402

LAURINI, R, THOMSON, D, 1992, *Fundamentals of Spatial Information Systems*, Academic Press, London

LEE, J, 2001, 3D Data Model for Representing Topological Relations of Urban Features, in *Proceeding of 21st Annual ESRI International User Conference*, San Diego, CA, [online], Available from http://gis.esri.com/library/userconf/proc01/professional/papers/pap565/p565.htm, [Accessed 12 December 2004]

LEVY, B, CAUMON, G, CONREAUX, S, CAVIN, X, 2001, Circular Incident Edge Lists: a Data Structure for Rendering Complex Unstructured Grids, *Proceedings of the conference on Visualization,* San Diego, California, [online], Available from http://gocad.ensg.inpl-nancy.fr/IMG/papers/2001/2001_ciel.levy.pdf, [Accessed 12 December 2004]

LI, S, 2006, A Complete Classification of Topological Relations using 9-Intersection Method, *International Journal of Geographical Information Science*, 20, 6

LI, S, 2006b, Combining Topological and Directional Information: First Results, in VELOSO, M, ed., *Proceedings of the International Joint Conference on Artificial Intelligence,* Lecture Notes in Computer Science, 4092, 252-264

LI, Z, ZHAO R, CHEN J, 2002, A Voronoi-Based Spatial Algebra for Spatial Relations, *Progress in Natural Science*, 12, 7

LIBKIN L, 2005, *Query Transaction Processing,* Lecture notes from Introduction to Databases Course, Department of Computer Science, University of Toronto, [online], Available from http://www.cs.toronto.edu/~libkin/csc343/fo3/set5.2up.pdf, [Accessed 5th September 2006]

LIN, T, WARD, M, POWER, L, LANDY D, 1995, From Databases to Visualisation – Providing a User Friendly Visual Environment for Creating 3D Solid Geology Models, in Proceedings of Application of Computers and Operations Research in the Minerals Industries, (APCOM), 11-20, [online], Available from http://web.cs.wpi.edu/~matt/research/pubs/sectionstar3_1.html, [Accessed 27th September 2006]

LIN, X, LIU, L, YUAN, Y, 2006, Summarizing Level-Two Topological Relations in Large Spatial Databases, *ACM Transactions on Database Systems,* 31 (2), 584-630

LIU, K, SHI, W, 2003, Analysis of topological relationships between two sets, in SHI, W, FISHER, P, GOODCHILD, M, eds., *Spatial Data Quality*, Taylor and Francis, London, 61-71. Cited in: ZLATANOVA, S, RAHMAN, A, SHI, W, 2004, Topological Models and Frameworks for 3D Spatial Objects, *Computers and Geosciences*, 30(4), 419-428

LIXIN, W, WENZHONG, S, 2003, GTP-Based Integral Real-3D Spatial Model for Engineering Excavation GIS (E2GIS), in *Publications of the Asia GIS Conference (AGISA)*, Session 13

MANNINGS, R, PARKER C, 2006, *The Invisible GIS – Technology Convergence to Make future GI user friendly*, [online], Available from http://www.ordnancesurvey.co.uk/oswebsite/partnerships/research/publications/docs/2006/mannings_parker_foresight.pdf, [Accessed 20th October 2006]

MAPINFO, 2006, *MapInfo Professional – Overview*, [online], Available from http://extranet.mapinfo.com/products/Overview.cfm?productid=1044&productcategoryid=1, [Accessed 20th October 2006]

MARK, D, EGENHOFER, M, 1994, Calibrating the Meanings of Spatial Predicates From Natural Language: Line-region Relations. In *Proceedings, Spatial Data Handling*, 1, 538-553

MARK, D, EGENHOFER, M, 1995, Topology of Prototypical Spatial Relations between Lines and Regions in English and Spanish, in PEUQUET, D, ed., *Proceedings of AutoCarto12*, North Carolina, 245-254

MARTIN, A, 2000, The ups and downs of protein topology; rapid comparison of protein structures, *Protein Engineering* 13(12), 829-837

MASINI A, 2006, *Technology Strategy Course Notes*, London Business School

MCCANN, M, 2002, Creating 3D Oceanographic Data Visualizations for the Web, *Proceedings of the seventh international conference on 3D Web Technology*, Arizona, USA, 179-184

MCDONNELL, R, KEMP, K 1996 *International GIS Dictionary*, Chichester, Wiley Europe

MOLENAAR, M, 1990, A formal data structure for three-dimensional vector maps. In BRASCEL, R, KISOMOTO, H, eds., *Proceedings of the Fourth International Symposium on Spatial Data Handling, Zurich, Switzerland*, 830-843.

MOLENAAR, M, 1992, A Topology for 3D Vector Maps, in STEWART A, ed., ITC Journal, 1, 25-33

MONTGOMERY, G, SCHUCH, H, 1993, GIS Data Conversion Handbook Wiley, USA

NANDA, S, 2003, Use of Spatial/3D Position Information for Location Aware Routing in Ad-hoc Wireless Networks, [online], Available from http://www.cs.dartmouth.edu/~snanda/presentations/Thesis_proposal_Nanda.doc, [Accessed 12 December 2004]

NEBIKER, S, 2003, Support for Visualisation and Animation in a Scalable 3D GIS Environment – Motivation, Concepts and Implementation. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Volume XXXIV-5/W10

NGUYEN, V, PARENT, C, SPACCAPIETRA, S, 1997, Complex Regions in Topological Queries, in *Proceedings of the Conference on Spatial Information Theory (COSIT)*, Lecture Notes in Computer Science, 1329, 175-192

OGC, 2006, Open Geospatial Consortium – Open GIS Specifications (Standards), [online], Available from http://www.opengeospatial.org/docs/01-101.pdf, [Accessed 12 April 2006]

ONSRUD, H, 2003, Making a Cadastre law for 3D properties in Norway, *Computers, Environment and Urban Systems* 27, 375-382

ORACLE, 2002, *Overview of SQL Statement Processing – Doc ID Note: 199273.1* [online], Available from https://metalink.oracle.com/metalink/plsql/f?p=130:14:5420866989688716 70, [Accessed 12th September 2006]

ORACLE, 2006, *The Oracle Database*, [online] Available from http://www.oracle.com/database/index.html, [Accessed 15th November 2006]

ORACLE, 2006a, *Oracle Spatial & Oracle Locator: Location Features for Oracle Database 10g*, [online], Available from http://www.oracle.com/technology/products/spatial/index.html, [Accessed 16 May 2006]

ORACLE, 2006b, *Using Explain Plan,* [online], Available from http://download-east.oracle.com/docs/cd/B19306_01/server.102/b14211/ex_plan.htm, [Accessed 12th September 2006]

ORACLE, 2006c, *The Query Optimizer,* [online], Available from http://download-east.oracle.com/docs/cd/B19306_01/server.102/b14211/optimops.htm, [Accessed 12th August 2006]

ORACLE, 2006d, *Oracle Spatial User's Guide and Reference,* [online]. Available from http://download-east.oracle.com/docs/cd/B19306_01/appdev.102/b14255/sdo_operat.htm, [Accessed 8th September 2006]

ORACLE, 2006e, *SQL Tuning Overview,* [online]. Available from http://download-east.oracle.com/docs/cd/B19306_01/server.102/b14211/sql_1016.htm#sthref1062. [Accessed 9th September 2006]

ORACLE, 2006f, *Initialisation Parameters,* [online], Available from http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14237/initparams_part.htm, [Accessed 12th August 2006]

ORACLE, 2006g, *The Analyze Statement,* [online], Available from http://download-uk.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_4005.htm, [Accessed 12th August 2006]

ORACLE, 2007, Oracle Spatial Technologies: Oracle Maps and 11g Planned Features [online], Available from http://www.oracle.com/technology/products/spatial/pdf/oow2006_ioug_presentations/dlapp_oow2006.pdf [Accessed 12th February 2007]

ORACLE, 2007b, Oracle® Spatial Topology and Network Data Models, [online], Available from: http://download-west.oracle.com/docs/cd/B19306_01/appdev.102/b14256/toc.htm, [Accessed 12th March 2007]

ORACLE, 2007c, Introduction to Oracle Objects, [online], Available from: http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28371/adobjint.htm#sthref80, [Accessed 21 September 2007]

ORACLE, 2007d, Designing and Developing for Performance, [online], Available from: http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10752/design.htm, [Accessed 21 September 2007]

ORDNANCE SURVEY, 2004, Ordnance Survey Feature Extraction Working Group – Automatic 3D Representation of Street Furniture, [online], Available from http://www.ordnancesurvey.co.uk/education/pdf/streetfurniture.pdf [Accessed 12 December 2004]

ORDNANCE SURVEY, 2006, Ordnance Survey Products – Welcome to OS MasterMap, [online], Available from http://www.ordnancesurvey.co.uk/oswebsite/education/pdf/streetfurniture.pdf, [Accessed 23rd December 2006]

PAPADIAS, D, THEODORIDIS, Y, 1997, Spatial Relations, Minimum Bounding Rectangles, and Spatial Data Structures, *International Journal of Geographical Information Science*, 11(2), 111-138

PARASOLID, 2007, *Parasolid Bodyshop – Key Features*, [online], Available from: http://www.ugs.com/products/open/parasolid/portfolio/bodyshop.shtml, [Accessed 23rd September 2007]

PARK S-H, RYU, K, GILBERT, D, 2005, Fast Similarity Search for Protein 3D Structures using Topological Pattern Matching Based on Spatial Relations, *International Journal of Neural Systems,* 15 (4), 287-296

PARKER, C, 2004, Research Challenges for a Geo-Information Business, *The Cartographic Journal - International Geographic Congress Special Issue*, 41 (2)

PENNINGA, F, 2005, 3D Topographic Data Modelling: Why Rigidity Is Preferable to Pragmatism, in COHN, A, MARK, D (eds.)., Anthony G. Cohn and David M. Mark (Eds.), *Spatial Information Theory*, COSIT 2005, Ellicotville, 409-425

PENNINGA, F, 2006, Poincaré Simplicial Homology for 3D Volume Modelling Presented at the GIS Lunch Seminar at Delft University, September 1st Delft, [online], Available from http://www.frisopenninga.nl/, [Accessed 23rd September 2007]

PENNINGA, F, VAN OOSTEROM, P, KAZAR, B, 2006, A TEN-based DBMS approach for 3D Topographic Data Modelling, in: RIEDL, A. KAINZ, W, ELMES, G, (eds.). *Progress in Spatial Data Handling*, 12th International Symposium on Spatial Data Handling, 581-598

PENNINGA F, VAN OOSTEROM, P, 2007, A Compact Topological DBMS Data Structure For 3D Topography In: S.I. Fabrikant and M. Wachowicz (Eds.) *The European Information Society - Leading the Way with Geo-Information*, Lecture Notes in Geoinformation and Cartography, Springer, 455-471

PETERS, T, ROSEN, D, DORNEY, S, 1994, The Diversity of Topological Applications within Computer Aided Geometric Design, *Annals of the New York Academy of Sciences*, 728, 198-209

PFUND, M., 2001, Topologic data structure for a 3D GIS, Proceedings of the 3rd ISPRS Workshop on Dynamic and Multidimensional GIS, Bangkok, Thailand, 34 (2W2), 233-237

PIGOT, S, 1995, *A Topological Model for a 3-Dimensional Spatial Information System*, PhD Thesis, University of Tasmania, Australia

PILOUK, M, 1996, *Integrated modelling for 3D GIS*, PhD Thesis, ITC, The Netherlands.

POSTGIS, 2007, PostGIS Manual, [online], Available from http://postgis.refractions.net/docs/postgis.pdf, [Accessed 15th January 2007]

PREPARATA, P, SHAMOS, M, 1985, Computational Geometry: An Introduction, Springer-Verlag, New York

PRICE, R, TRYFONA N, JENSEN,C, 2001, Modeling Topological Constraints in Spatial Part-Whole Relationships, in KUNII, H, JAJODIA, S, SOLVBERG, A, eds., *Lecture Notes in Computer Science*, 2224, 27-40

RAMOS, F, 2002, A multi-level approach for 3D modelling in Geographical Information Systems, in *ISPRS Commission IV Symposium on Geospatial Theory, Processing and Applications*, Ottowa, Canada

REICHARDT, M, 2004, *The Havoc of Non-Interoperability, OGC White Paper* , [online], Available From http://portal.opengeospatial.org/files/?artifact_id=5097, [Accessed 17th March 2007]

RIGAUX, P, SCHOLL, M, VOISARD, A, 2002, *Spatial Databases: With Application to GIS*, Morgan Kauffman, London

RIKKERS, R, MOLENAAR, M, STUIVER, J, 1994, A query oriented implementation of a topologic data structure for 3-dimensional vector maps, *International Journal of Geographical Information Systems*, 8(3), 243 – 260

ROSSIGNAC, J, 1996, Specification, representation, and construction of non-manifold geometric structures, *ACM SIGGRAPH [online], Available from: http://www.gvu.gatech.edu/~jarek/papers/SIG96sgc.pdf , [Accessed 23rd September 2007]*

ROWE, J, RAZDAN, A, SIMON, A, 2003, Acquisition, Representation, Query and Analysis of Spatial Data: A Demonstration 3D Digital Library, *Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital libraries*, Houston, Texas, 147-158

ROY, C, 2005, *A Java 3D Visualisation Tool for Oracle Spatial SDO_GEOMETRY*, M Sc. Project Report, University College London

SAMET, H, 1995, Spatial data Structures, in KIM, W, ed., *Modern Database Systems: The Object Model, Interoperability and Beyond*, Addison-Wesley/ACM Press, 361-385.

SCALZO, B, 2006, *Engineering Better PL/SQL*,[online], Available from: http://www.orafaq.com/node/846, [Accessed 8th September 2006]

SCHNEIDER, M, BEHR, T, 2006, Topological Relationships Between Complex Spatial Objects, *ACM Transactions on Database Systems*, 31 (1)

SCHNEIDER, P, EBERLY, D, 2003, Geometric Tools for Computer Graphics (The Morgan Kaufmann Series in Computer Graphics), Morgan Kaufmann, San Francisco

SHARIFF A, EGENHOFER, M, MARK, D, 1998, Natural-Language Spatial Relations between Linear and Areal Objects: The Topology and Metric of English Language Terms, *International Journal of Geographical Information Science*, 12 (3), 215-246

SHARMA, J, 1996, *Integrated Spatial reasoning in Geographic Information Systems: Combining topology and Direction*, PhD thesis, University of Maine

SHI, W, GUO, W, 2002, Topological relationships between spatial objects with uncertainty, in SHI, W, FISHER, P, GOODCHILD, M, eds., *Spatial Data Quality*, Taylor and Francis, London, 50-61. Cited in: ZLATANOVA, S, RAHMAN, A, SHI, W, 2004, Topological Models and Frameworks for 3D Spatial Objects, *Computers and Geosciences*, 30(4), 419-428

SHI, W, LIU K, 2005, Are Topological Relations Dependent on the Shape of Spatial Objects, *Progress in Natural Science*, 15(11)

SHI, W, YANG, B, LI, Q, 2003, An object-oriented data model for complex objects in three-dimensional geographic information systems, *International Journal of Geographical Information Science*, 17(5), 411-430

SILBERSCHATZ, A, KORTH,K, SUDARSHAN, S, 2002, *Database System Concepts – 4th Edition*, McGraw-Hill Higher Education, New York

SILVA, G, GOMES, A, 2005, Oversimplified Euler Operators for a Non-Oriented, Non-Manifold B-Rep Data Structure, in BEBIS, G, BOYLE, R, KORACIN, D, BAHRAM P, eds., *Advances in Visual Computing: First International*, Lecture Notes in Computer Science, 3804, 25-34

SILVER, D, CARSTEN, J, THAYER, S, 2005, Topological Global Localization for Subterranean Voids, in Field and Service Robotics, Springer Tracts in Advanced Robotics, 25, 117-128

SIRAKOV, N, RIBEIRO, L, PINA, P, MUGE, F, 2000, An Algorithm for 3D Groundwater Units Reconstruction and Visualization. In KOVAR, K, VAN DER HEIJDE, P, eds., *Calibration and Reliability in Groundwater Modeling* International Association of Hydrological Sciences (IAHS) Press, Publication 265, 148-154.

SMITH, G, 2006, What's Your Excuse, *Directions Magazine*, [online], Available from: http://www.directionsmag.com/article.php?article_id=2300, [Accessed 21st March 2007]

SPIKINS, P, CONNELLER, C, AYESTARAN, H, SCAIFE, B, 2002, GIS Based Interpolation Applied to Distinguishing Occupation Phases of Early Prehistoric Sites, *Journal of Archaeological Science*, 29, 1235-1245

STOTER, J, SALZMANN, M, 2003, Towards a 3D Cadastre: where do cadastral needs and technical possibilities meet?, *Computers, Environment and Urban Systems*, 27, 395-410

SUN, C, AGRAWAL, D, EL ABBADI, A, 2002 Exploring Spatial Datasets with Histograms, *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, IEEE Computer Society, 93 – 102

SZYMCZAK, A, TANNENBAUM A, MISCHAIKOW, K, 2006, Coronary Vessel cores from 3D Imagery: a topological approach, *Medical Image Analysis*, 10(4), 548-559

TAKINO, S, 2000, Topological Network Model for Improving Urban 3D Data Use, in *International Workshop on Urban 3D/Multi-Media Mapping* University of Tokyo, Japan, [online], Available from http://www.dawn-corp.co.jp/randd/UM32000.pdf, [Accessed 12 December 2004]

TEMPFLI, K, 1998, 3D Topographic Mapping for Urban GIS, *ITC Journal*, 3/4, 181-190

THEOBALD, D, 2001, Topology revisited: representing spatial relations, *International Journal of Geographical Information Science*, 15, 689-705

THOMPSON, R, 2005, Proofs of Assertions in the Investigation of the Regular Polytope, *GISt Report No. 41/NRM-ISS090*, Delft University of Technology (TU Delft)

THOMPSON, R, van OOSTEROM, P, PULLAR P, 2006, Robust Representation and Analysis of Geo-information, Proceedings of AutoCarto 2006, Vancouver USA

TOAD, 2006, *Product Features*, [online], Available from: http://toadsoft.com/feats.html, [Accessed 12th September 2006]

TSE, C, 2006, *3D GIS Java Visualisation Tool*, B. Eng. Project Report, University College London

TSE, R, GOLD, C, 2003, A proposed connectivity-based model for a 3-D Cadastre, *Computers, Environment and Urban Systems*, 27, 427-445

VAN DER MOLEN, P, 2003, Institutional Aspects of 3D Cadastres, *Computers, Environment and Urban Systems*, 27, 383-394

VAN DER MOST, A, 2004, *An Algorithm for Overlaying 3D Features Using a Tetrahedral Network* Master's Thesis TU Delft, 2004, 96 p

VAN OOSTEROM, P, PLOEGER, H, STOTER, J, THOMSON, R, LEMMEN, C, 2006, Aspects of a 4D Cadastre: A First Exploration, *Proceedings of the XXXII International FIG Congress*, Munich

VAN OOSTEROM, P, VERTEGAAL, W, VAN HEKKEN, M, 1994, Integrated 3D Modelling within a GIS, *Proceedings of Advanced Geographic Data Modelling (AGDM),* Delft, The Netherlands

VAN SMAALEN, J, 2003, Automated Aggregation of Geographic Objects, A New Approach to the Conceptual Generalisation of Geographic Databases, Doctoral Dissertation, Wageningen University, The Netherlands

VIDELA, E, KNOX-ROBINSON, C, 1997, Three Dimensional Computer-Based Gold Prospectivity Mapping using Conventional Geographic Information Systems, Three Dimensional Mine Visualization Software and Custom-Built Spatial Analysis Tools, [online], Available from http://www.geocomputation.org/1997/papers/pvidela.pdf, [Accessed 12 December 2004]

WANG, X, GRUEN, A, 1999, The Configuration and the Implementation of a hybrid 3-D GIS for Urban Data Management, *Geographic Information Science,* 4(1-2)

WARE, J, JONES, C, 1998, Matching and Aligning Features in Overlayed Coverages, in *Proc. of 6th International Symposium on Advances in Geographical Information Systems* (ACM-GIS'98), 28-33

WEI, G, 1996, Three-Dimensional Representation of Spatial Object and Topological Relationships, *International Archives of Photogrammetry and Remote Sensing*, XXXI (B3)

WEI, G, PING, Z, JUN, C, 1998, Topological Data Modelling for 3D GIS, *The International Archives of Photogrammetry and Remote Sensing,* XXXII (4)

WHITE, M, 1984, Technical Requirements and Standards for a Multipurpose Geographic Data, cited in GALDI, D, 2005, Spatial Data Storage and the Topology in the Redesigned MAF/TIGER System , US Census Bureau, [online], Available from http://www.census.gov/geo/mtep_obj2/topo_and_data_stor.html, [Accessed 16th April 2006]

WINTER S, 1995, Topological Relations between Discrete Regions, in *Advances of Spatial Databases, Proc. Fourth Symposium on Large Spatial Databases (SSD 95)*, Maine, Lecture Notes in Computer Science 951, 310-328.

WORBOYS M, 1995, *GIS – A Computing Perspective.* London, Taylor & Francis

WORBOYS, M, DUCKHAM, M, 2004, *GIS A Computing Perspective* (Second Edition), CRC Press

WUST, T, NEBIKER, S, LANDOLT, R, 2004 Applying the 3D GIS DILAS to Archaeology and Cultural Heritage Projects – Requirements and First Results, *International Archives of Photogrammetry and Remote Sensing and Spatial Information Sciences*, 35 (5), 407-412

YE H, KERHERVE B, VON BOCHMANN G, BOURNE D, 2002, Towards Database Scalability Through Efficient Data Distribution in E-Commerce Environments, in WILLIAMS, A, ed., *Proceedings of the Third International Symposium on Electronic Commerce*, IEEE Computer Society, 87-95

ZEITOUNI, K, DE CAMBRAY, B, DELPY, P, 1995, Topological Modelling for 3D GIS, In WYATT, R, HEMAYET H, eds., *Fourth International Conference on Computers in Urban Planning and Urban Management*

ZHAO, R, CHEN, J, 1999, Enhancing V9i Model for Spatial Relationships with Voronoi Distance, in HUAND, G, CHEN, Y, eds., Towards Digital Earth: Proceedings of the International Symposium on Digital Earth, Science Press

ZHOU, X, CHEN J, LI Z, ZHAO R, ZHU, J, 2005, A Model for Topological Relationships based on Euler Number, *Proceedings of the ISPRS Hangzhou 2005 Workshop*, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences Volume XXXVI, Part 4/W6

ZLATANOVA, S, 2000, *3D GIS for Urban Development,* ITC Dissertation Series No. 69

ZLATANOVA, S, HOLWEG, D, COORS, V, 2004, Geometrical and Topological Models for Real-time GIS, in FENDEL, E, RUMOR, M, eds., *Proceedings of the 24th Urban Data Management Symposium*, Chioggia, October 27-29, 3.33 – 3.46

ZLATANOVA, S, RAHMAN, A, PILOUK, K, 2002, 3D GIS: Current Status and Perspectives, in *Proceedings of the ISPRS Commission IV Symposium, Geospatial Theory, Processing and Applications International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, GITC, Lemmer, The Netherlands, 34 (94), 216-221.

ZLATANOVA, S, RAHMAN, A, SHI, W, 2004, Topological Models and Frameworks for 3D Spatial Objects, *Computers and Geosciences*, 30(4), 419-428

Appendix 1 – Using Topology for Data Quality Improvement

The process of populating and maintaining data in a topological structure such as 3DFDS has the additional benefit of validating the correctness of the data itself, and allowing errors to be identified and corrected. These include errors such as Faces not forming a closed shell (leaving, for example, a building with a gap between two walls) or the presence of undershoots (polygons not closed) or overlaps within the dataset.

Validation is required to ensure that topological relationships between objects are correctly identified and that data is otherwise generally high quality and fit for purpose. A number of algorithms that can be used to support data validation are described here in relation to datasets where valid objects are required to be manifold and have planar faces. Note that these are provide as examples of the processes that would be included in the Topological Engine (beyond the scope of this thesis) and do not represent a complete validation process. For example, algorithms the detection of overshoots, undershoots and self intersecting lines or surfaces are not listed.

However, it should be noted that data validation processes for STS will vary depending on the nature of the data being modelled (as described in Section 5.7).

Validating Topological Consistency of Simple Objects

The following equation, known as Euler's Formula (Laurini and Thomson 1992), can be used for object validation purposes, and validates that a polyhedron is simple (i.e. is topologically equivalent - can be continuously deformed - to a sphere). This ensures that the Faces do in face enclose a 3D object:

$$V + F = E + S + P \qquad \textbf{Euler Equation}$$

Where V is the number of vertices (Nodes) in a graph, F is the number of Faces E is the number of Edges and P is the number of sub graphs (i.e. graphs whose vertices form a subset of the main graph). S is the Euler number, which provides the balance for the equation. This number varies depending on whether the outside region of the graph is also taken to be a Face. If the outside region is a Face, then S = 2 otherwise S = 1.

The Euler formula provides a method for validating the construction of single geometric objects i.e. ensuring that they are closed and orientable (and hence not non-manifold) (Silva and Gomes 2005). Manifold objects are required to support the automatic determination of the interior and boundary of objects, and hence topological relationships between such objects. Although this

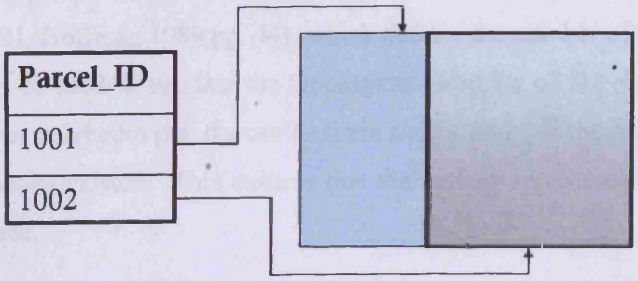
formula is based on graph theory, this does not preclude its application in situations where other data structures are used, provided that the Vertices, Edges, Faces, sub-graphs and S for the object can be identified.

Planar Enforcement and Intersection

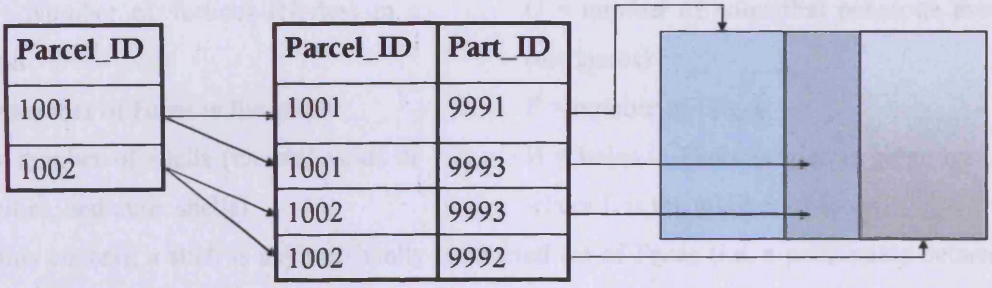
The above rules apply to planar graphs, thus imposing a requirement for planarity onto 2D topological structures. In this situation, vendors therefore utilise a concept of planar enforcement to apply this rule ensuring that the topological cells (primitives) making up the geometric objects cannot overlap.

Planar enforcement ensures that no two topological cells can occupy the same space at the same time. In other words (Goodchild, 1990) planar enforcement is the term used to refer to the rules used in converting the standard representation of reality (where multiple objects occupy the same space) into a single-valued function defined everywhere. Planar enforcement occurs when converting/improving quality of spaghetti data or during spatial interpolation, where it is used to assign values to every point in the plane.

The process of planar enforcement does not allow overlapping objects to be modelled. It is therefore necessary to break down overlapping objects into constituent parts and then determine if any objects share common components of topology and therefore intersect. The results of this process are shown in Figure 105 below.



Geometrical Data Structure – Prior to Planar Enforcement



Topological Data Structure – Post Planar Enforcement

Figure 105 - Additional Complexity added by Planar Enforcement

Validating Topological Consistency of Polyhedra with Cavities

De La Losa and Cervelle (1999) note that the majority of 2D topological models are based on planar graph theory, which allows network analysis to be performed. This approach cannot be extended into 3D as in 3D there is a requirement to model more than two Faces adjacent to the same Edge - not possible in a planar graph. A concept similar to planar enforcement, which can be termed volumar enforcement, is still useful in the context of the 3D situation. This can be defined as ensuring that no two topological primitives exist at the same point in space. The resulting deconstruction of the 3D objects is shown in Figure 106 below.

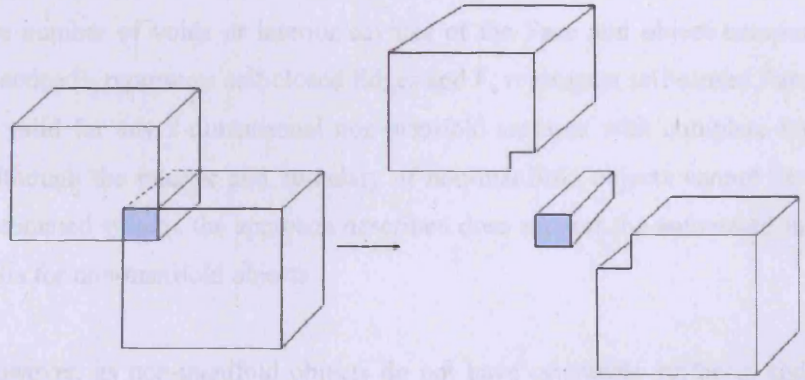


Figure 106 - Intersection in 3D - Volumar Enforcement

The Euler equation described above has been extended to the Euler-Poincare formula (Lee, 2001, Hoffman 1989 pg. 44), which defines the number of Edges and Faces of a manifold. This can be used to validate the topological structure of 3D objects with cavities, ensuring that the inner polyhedra (i.e. the cavities) are simple and that the outer polyhedron would be simple if no cavities existed. This ensures that the surface representation does in fact enclose a complete, space.

$$V - E + F - H - 2(S-G) = 0$$ **Euler Poincare Formula**

Where:

V = Number of vertices (Nodes) in a graph

F = number of Faces in the graph

S = number of shells (internal voids or cavities, and outer shells)

G = number of holes that penetrate the solid (the genus)

E = number of Edges

H = holes in Faces (which is given by L – F, where L is the number of loops)

In this context, a shell is any maximally connected set of Faces (i.e. a path exists between all vertices), and a loop is any maximal closed chain of Edges on the boundary of a Face (DeFloriani, 2003). Penninga (2006) notes that Euler Poincare holds for all simplicial complexes, including those with dangling Edges and Faces – i.e. this is a necessary but not sufficient test for object validity.

Validation of Non-Manifold Objects

Object validation as described above is applicable to manifold objects. An extension of the formula can also be identified for non-manifold objects (Silva and Gomes 2005). This is given by:

$$V - (E-E_h) + (F-F_h + F_c) = C - C_h + C_c$$ **Equation 2 - Euler Formula for Non-Manifold Objects**

where F, E and V denote the number of Faces, Edges and Vertices, and E_h, F_h and C_h denote the number of holes or handles of an Edge or a Face or an object component (C). F_c and C_c are the number of voids or interior cavities of the Face and object components respectively. In practice E_h represents self-closed Edges and F_c represents self-closed Faces. This Euler formula is valid for any 2-dimenisonal non-manifold surfaces with complete boundaries in 3D space. Although the interior and boundary of non-manifold objects cannot be easily identified in an automated system, the approach described does support the automated insertion and removal of cells for non-manifold objects.

However, as non-manifold objects do not have orientable surfaces, and making it difficult to distinguish the interior and exterior, it is less likely that this approach will be used for validation in the context of binary topological relationships.

Validating Face Planarity

This can be determined in two steps:

1. Identify the equation of the plane formed by three Nodes on the boundary of the Face. This is given by a Node N_0 on the Plane and the equation of the Normal to the plane (using the cross-product):

   $\vec{n} = (V1 - V3) \times (V2 - V3)$ **Cross Product to define a Normal Vector**

2. Measure the distance from all other Nodes N_i to the resulting plane. This should be 0 (allowing for errors caused by the implementation of these algorithms in a computerised environment – see Section **Error! Reference source not found.**):
   - S1 = dot product of the Normal Vector to the Plane and the vector between N_i and N_0
   - S2= dot product of the Normal Vector to the Plane with itself
   - S3 = S1/S2
   - B = N_i + S3* (Normal Vector) (this is the vector of the point where N_i touches the plane)
   - D = distance between B and N_i

D should be 0 or within a given tolerance for all Nodes.

S= ((V1.x * V2.x)+ (V1.y * V2.y) + (V1.z * V2.z)) **Dot Product of two Vectors V1 and V2**

$\vec{n}$ = (V1.y * V2.z – V1.z * V2.y) **x** + (V1.z * V2.x – V1.x * V2.z) **y** + (V1.x * V2.y – V1.y * V2.x) **z** **Cross Product equation between V1 and V2**

Validating that an Object is Closed

1. This depends on the dimension of the object in question:
   a. If Dimension = 3, then all Edge primitives should be associated with two and only two Faces
   b. If Dimension = 2 then all Nodes should be associated with two and only two Edges

Appendix 2 – Topological Frameworks

A brief description of the topological frameworks described in Chapter 2 is given here.

The 9-Intersection Framework

One of the first formal topological frameworks developed was that by Egenhofer and Franzosa (1991), namely the 4-intersection model, which examines the intersections between the boundaries and interiors of objects and categorises the intersections according to whether common elements exist. A matrix can be drawn up identifying all the possible intersection options between the two objects. This has been extended by Egenhofer and Herring (1990) to the 9-intersection model, which identifies a total of 512 theoretically possible binary relations in 2D Euclidean Space (R^2), although many of these cannot occur in practice.

Using standard notation, with objects A and B, the possible relations can therefore be shown in the matrix below

$$R(A,B) = \begin{pmatrix} Int(A) \cap Int(B) & Int(A) \cap Bnd(B) & Int(A) \cap Ext(B) \\ Bnd(A) \cap Int(B) & Bnd(A) \cap Bnd(B) & Bnd(A) \cap Ext(B) \\ Ext(A) \cap Int(B) & Ext(A) \cap Bnd(B) & Ext(A) \cap Ext(B) \end{pmatrix}$$

(Taken from Egenhofer and Herring 1990)

In the above, Int(A) represents the interior of object A, Bnd(A) represents the boundary of A and Ext(A) represents the exterior of A. The symbol ∩ represents the intersection operator.

Each intersection operation can return a non-empty result (for example the boundary of A intersects with the boundary of B) or an empty result. There are a total of 2^9 (512) possible results. Conditions to determine valid 9-Intersections relationships are given by Egenhofer and Herring (1990) for 2D and by Zlatanova (2000) for the 3D case.

Dimensional Model

The Dimensional Model (Billen *et al.,* 2002) takes a different approach to relationship identification, examining at the Order of objects. The authors use the concept of a hyper plane to define Order as the dimension of the intersection of all supporting hyper planes, and extend this to define an order formula for manifolds (to support non-convex objects). The dimensional model defines both the spatial objects and the relationships between the objects – the objects are composed of dimensional elements, and the Dimensional Model handles relationships between these. Definitions of extension (consisting of all points of order n where n is the dimension of the object) and limit (all points of order 0 to n-1) are presented.

Three types of relationships are identified (Billen and Zlatanova 2003) – total (the intersection between two dimensional elements is equal to the first element, and the intersection between their extensions is not empty), partial (the intersection is not equal to the first dimensional element, but the intersection between their extensions is not empty) or no-relation (the intersection between their extensions is empty). Using the total, partial and no-relation concepts, a series of basic, extended and simplified relationships are defined. Basic relationships examine the relationships between each dimensional element of the two objects and assign 0 for non-existent, 1 for total and 2 for partial. Extended relationships reuse this concept, but also look at the dimension of the intersection itself. Finally, simplified relationships remove any elements that are not geographically relevant from the list of dimensional elements (for example, the 0D element may not be relevant in a relationship between a 3D cube and a 2D polygon).

Voronoi 9-Intersection

As with the 9-Intersection framework described above, this framework builds up topological relationships by identifying the intersections between the interior, boundary and exterior of objects. However, Chen *et al.* (2001) identify an issue with the definition of exterior in the 9-Intersection model (which in the Egenhofer and Herring (1990) approach is taken to be the complement of the interior and the boundary) when the co-dimension of an object is not 0. For example when embedding a line in 2-dimensional space (i.e. co-dimension 1), the boundary (defined as the end Nodes of the line) does not separate the interior from the embedding space. They therefore propose an alternative definition of the exterior, derived from the Voronoi region for each object. They argue that algorithms to identify the Voronoi regions for geometric objects are well established and that this approach overcomes issues with the 9-intersection method. Other authors have explored this approach further, including Zhao and Chen (1999).

The Chen *et al.* (2001) definition of exterior precludes the conditions-based approach to identification of topological situations that can exist in practice – as it can no longer be argued that the exteriors of any two objects will always interact, which is taken as given for standard 9-Intersection. This more restricted definition of exterior allows for additional relationship identification – in particular, that between complex entities with holes. Chen *et al.* (2001) have not explored extending the Voronoi 9-intersection framework into 3D, where Voronoi creation processes are less well defined.

Dimension Extended Model

The Dimension-Extended model was defined by Clementini *et al.* (1993). Again, this is based on the identification of intersections between the interior, boundary and exterior of objects,

although Clementini *et al.* (1993) also use the dimension of each intersection to distinguish between various topological relationships. They summarise the Dimension-Extended relationships using a Calculus-based approach, and define five key relationships, namely Touch, In, Cross, Overlap and Disjoint. A similar approach is described by Van Oosterom *et al.* (1994) for 3D, who note that the original definition of Cross in 2D does not hold for 3D situations, as in 3D the crossing objects may not always be lines, and propose an alternate definition to allow for this:

Relationship	*Criteria*	*Description*
Touch	$(A^0 \cap B^0 = \phi)\text{^}(A \cap B = 0)$	The interiors of A and B do not intersect, but A and B intersect in some way
In	$(A^0 \cap B^0 = \phi)\text{^}(A \cap B = A)$	The interiors of A and B do not intersect, but intersecting A and B returns A
Cross	$\dim(A^0 \cap B^0 < (\max(\dim(A^0), \dim(B^0))$ $\text{^}(A \cap B \neq A)\text{^}(A \cap B) \neq B)$	The dimension of the intersection between A and B is the maximum of the dimensions of A and B, and A intersect B does not return A and A intersect B does not return B
Overlap	$(\dim(A^0 \cap B^0) = \dim(A^0) = \dim(B^0))$ $\text{^}(A \cap B \neq A)\text{^}(A \cap B \neq B)$	The dimension of the intersections of the interiors of A and B is the same as the dimensions of the interior of A and the dimension of the interior of B, and A is not in B and B is not in A
Equal	$(AinB)\text{^}(BinA)$	A is in B and B is in A

Table 84 - Topological Relationships proposed by Wei *et al.* (1998)

Boundary/Co-Boundary Framework

Pigot (1995, pg. 111) proposes that relationships are identified by creating a third object by taking the union of the two objects under examination. The relationships within the cell neighbourhoods of the resulting object can then be identified using generic operators based on the boundary-co-boundary relationships. The Boundary of an n-cell consists of the n-1 cells incident to it. The co-boundary of an n-cell consists of the n+1 cells incident to it.

SBox

Similarly to Region Connection Calculus below, this framework, devised by Haarslev and Moller (1997) has been designed to support qualitative reasoning – i.e. the development of techniques to enable a computer to reason about system behaviour without precise quantitative information.

The framework is based on the 9-Intersection framework, and utilises the same concepts of interior, boundary and exterior. However, a hierarchy of relationships is defined. The hierarchy first looks at objects as a whole (are they SPATIALLY_RELATED or DISJOINT) before differentiating between more detailed relationships on object parts. A series of generic terms is also defined to represent the relationships, as shown in Table 85 below:

Relationship Name	*Description*
SPATIALLY_RELATED	Defined as the disjunction between its two mutually exclusive sub-relations disjoint and connected.
DISJOINT	Two objects are disjoint if their intersection is empty.
CONNECTED	Two objects are connected if their intersection is non-empty.
BPPXG_OVERLAPPING	Two objects are generally overlapping – this is defined as the disjunction of its two mutually exclusive sub-relations touching and S_OVERLAPPING.
TOUCHING	Two objects are touching if only their boundaries are intersecting (i.e. the interiors do not intersect).
S_OVERLAPPING	Two objects are strictly overlapping if they are connected and their intersection is not equal to either of them.
G_CONTAINS/G_INSIDE	An object A generally contains B is defined as the disjunction of its three mutually exclusive sub-relations, EQUAL, T_CONTAINS and S_CONTAINS (G_INSIDE is the inverse of this).
EQUAL	A object A is equal to an object B if they are CONNECTED and their intersection is equal to both of them.
T_CONTAINS/T_INSIDE	An object A tangentially contains an object B if their intersection is equal to B and the intersection of their boundaries is non-empty. T_INSIDE is the inverse of T_CONTAINS.
S_CONTAINS/S_INSIDE	An object A strictly contains an object B if their intersection is equal to B and only the interiors intersect (i.e. no boundary intersection). S_INSIDE is the inverse of S_CONTAINS.

Table 85 - Relationships Identified by SBox

Region Connection Calculus

Region Connection Calculus (RCC) (Cohn *et al.* 1997) was developed to support the process of qualitative spatial reasoning, and to overcome issues when mapping point-set and algebraic topology to that described from an end-user's perspective. The authors argue that topology as practised by mathematicians gives rise to some counter-intuitive objects – for example, a disc without boundary covers the same area as a disc with a boundary (as the boundary is infinitely thin in theory). Also an open finite line segment (i.e. without end-points) is topologically equivalent to an infinite line but not topologically equivalent to a closed (i.e. with end points) line segment.

In RCC, the extended REGIONS of space are taken as primary (i.e. as the main primitive) rather than the dimensionless points of traditional GIS. RCC Regions may be of arbitrary dimension but may not be of mixed dimensions. They must also be regular and must not be null. In RCC it is not possible to distinguish between regions that are open or closed. The authors argue that

this distinction is not required as the regions occupy the same amount of space. Relationships distinguished by the framework include disconnected (DC), part of, proper part of, identical (EQ), overlap, discrete from, partially overlaps (PO), externally connected (EC), tangential proper part (TPP) and non-tangential proper part (NTPP). These are described in Table 86 below, where C(x, y) is the general Connected relationship.

Relationship Code	*Description*	*Properties*
DC(x, y)	x is disconnected from y	¬ C(x,y) (not C(x,y))
P(x,y)	x is a part of y	for all z[C(z,x) → C(z,y)]
PP(x,y)	x is a proper part of y	$P(x,y) \wedge \neg P(y,x)$
EQ(x,y)	x is identical with y	P(x,y) ^ P(y,x)
O(x,y)	x overlaps y	$\exists z[P(x,y) \wedge P(z,y)$
DR(x,y)	x is discrete from y	¬ O(x,y)
PO(x,y)	x partially overlaps y	$O(x,y) \wedge \neg P(x,y) \wedge \neg P(y,x)$
EC(x,y)	x is externally connected by y	C(x,y) ^¬ O(x,y)
TPP (x,y)	x is a tangential proper part of y	PP(x,y) ^ $\exists z[EC(z,x) \wedge EC(z,y)$
NTPP(x,y)	x is a non-tangential proper part of y	$PP(x,y) \wedge \neg\exists z[EC(z,x) \wedge EC(z,y)$

Table 86 - Relationships in Region Connected Calculus

These are illustrated as shown below for simple regions. These relationships correspond to those identified for simple regions by the 9-Intersection framework (Egenhofer and Herring 1990):

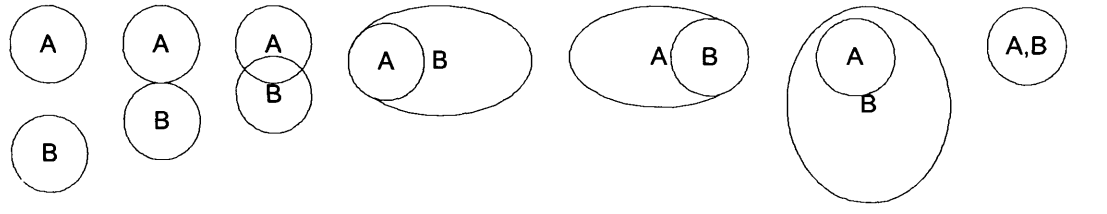


Figure 107 - Relationships between simple regions in RCC (from Cohn *et al.* 1997)

Once these relationships are defined, a Composition table can then be used to support the rapid identification of relationships of order greater than two. This is a pre-computed matrix containing information about the transitive closure of pairs of RCC relations - i.e. given 3 objects P,Q and R and two RCC relations C(P,Q) and C(Q,R) the matrix can be used to predict C(P,R).

Partially Ordered Sets

This framework (developed by Kainz *et al.* 1993) uses the concept of partially ordered sets (posets), which are binary relations (arbitrary associations of elements of one set with another or itself) which also meet the following conditions:

- reflexive (every element in the relation is related with itself)
- antisymmetric (if $x \leq y$ and $y \leq x$ then $x = y$)
- transitive (if $x \leq y$ and $y \leq z$, then $x \leq z$)

If P is a poset and $S \subseteq P$., an element $x \in P$ is an upper bound of S if $s \leq x$ for all $s \in S$. A similar definition applies to a lower bound. If the set of all upper bounds has a least element, then this is the least upper bound of S. If the set of all lower bounds has a greatest element, then this is the greatest lower bound of S. A lattice is a poset in which every pair of elements has a least upper bound and a greatest lower bound. A lattice is **complete** when a greatest lower bound and least upper bound exist for every subset of the poset. The process of normal completion specifies the number of elements that must be added to a poset to create a lattice.

The concepts of poset, upper bound, lower bound and lattice are then used to define a number of topological relationships, with definitions based on simplicial complexes. These are summarised in Table 87 below:

Relationship	*Definition*
Intersection	the intersection of the elements of set A is calculated as the greatest lower bound of A.
Boundary	The boundary of an area A can be determined by calculating the symmetric difference of the elements that are covered by all the triangles of A (symmetric difference is the set of elements belonging to one but not both of two given sets. It is therefore the union of the complement of A with respect to B and B with respect to A).
Neighbourhood	The neighbourhood of an area A can be defined as all the triangles that share a segment with A.
Touch	This can be derived from the set intersection which was calculated with the greatest lower bound. If the greatest lower bound is a triangle or set of triangles, then the areas INTERSECT. If the greatest lower bound is a segment or point then the areas TOUCH.
Containment	the upper bounds of a given set determine the containing areas.

Table 87 - Definitions and Relationships in the Poset framework

Following on from these definitions, relationship queries are mapped to the category of ordered sets. The authors also suggest that using the lattice approach can reduce complexity of some topological queries.

Euler Number Framework

This was developed by Zhou *et al.* (2005) following issues encountered with Voronoi-based algebra described by Chen *et al.* (2001). These include the fact that:

- It cannot handle tessellation-based objects – i.e. where coverage of the space is complete, such as land ownership parcels or topographic polygons. In these cases, the objects Voronoi region is itself, meaning that the equals and covers by relationships cannot be distinguished.
- It cannot handle time-based relationships (i.e. of objects that exist at different times).

Zhou *et al.* (2005) therefore suggest an alternative framework, in which they treat the spatial object as a whole object, and utilise the Euler number of the intersection components to enhance the description of these components. Additionally, they review set operators and conclude that only the intersection, difference or difference by operators are required to define a topological framework.

Zhou *et al.* (2005) first examine a simple matrix of values using content (does the intersection exist or not), dimension and number of connected components.

$$R(A,B) = \begin{bmatrix} A \cap B \\ A \setminus B \\ B \setminus A \end{bmatrix}$$ **Simple Relationship Matrix**

Each component of this matrix is associated with two values – the dimension of the shared object and the number of connected components. For example, for a polygon A inside polygon B, dimension (A∩B) = 2, number of connected components = 1. Dim (A\B) = -1, so the number of connected components is not calculated. Dim (B\A) = 2 and the number of connected components is 1. The resulting matrix is shown below.

$$R(A,B) = \begin{bmatrix} 21 \\ -1 \\ 21 \end{bmatrix}$$ **Equation 3 - Simple Relationship Matrix for Polygon Containment**

However, the number of connected components on its own is dependent on the shape of the objects in question. The authors therefore propose the Euler number (which represents the number of connected components minus the number of holes) as an alternative measure. They note that this approach uses simpler expressions to determine the relationships. The calculated Euler number can also be used to describe the shape of the intersection (no holes, one hole and so forth), enhancing the description of the relationship.

Object Shape Framework

This framework was developed by Shi and Liu (2005) and not only looks at the intersection of the interior and boundary components of objects, but also at the number of components of each intersection. The authors suggest that this is an important element of topology –while a circle and an ellipse are homeomorphic, the shared elements of the relationship between two circles may not be homeomorphic to that between two ellipses (for the circle, as a convex object, a maximum of one intersection component (Face or Edge) may result. For the ellipse, however, two disjoint Edges may be shared. These two Edges are never homeomorphic to the one Edge for the circle). Therefore the topological relationship is dependent on the shape of the object in question.

Part-Whole Framework

Developed by Price *et al.* (2001), this framework allows the specification of both binary and set-based topological constraints on composite spatial objects, and can be applied to binary or higher order relationships. It considers both the intersection and difference between two objects, determining three relationships for objects A and B, namely A∩B, A-B and B-A. A matrix of possible result combinations, describing high-level relationships, is drawn up. The

relationships are further sub-divided into sub categories, as shown in Table 88 below. Where boundary and interior are considered, the definition of boundary is application-specific, rather than being enforced by the framework itself.

A∩B	*A-B*	*B-A*	*First Level Relationship*	*Second Level Relationship*
Ø	Ø	Ø	No name. Occurs when both A and B are the empty set.	None
Ø	¬Ø	Ø	No name. Occurs when B is the empty set	None
Ø	Ø	¬Ø	No name. Occurs when A is the empty set.	None
Ø	¬Ø	¬Ø	Disjoint	Separate – minimum bounding figure (which is application dependent) does not intersect Interpenetrating – minimum bounding figure intersects
¬Ø	Ø	Ø	Equal	None
¬Ø	¬Ø	Ø	Contains (nested)	These are used when considering relationships between multiple objects Occasionally (A contains B. Some components of A do not contain B, and some components of B are not inside A. These components are in some other relationship with each other – i.e. not disjoint) Mostly (A contains B in every case tested. However, some components of A do not contain components of B) Completely (A contains B in every case tested and all components of B are inside A)
¬Ø	Ø	¬Ø	Inside (nested)	As for Contains
¬Ø	¬Ø	¬Ø	Connected	Boundary Overlap Interior Overlap Mixed Overlap

Table 88 - Relationships in the Part/Whole framework, adapted from Price *et al.* (2001)

Regular Polytope Framework

Thomson (2005) suggests an approach to object definition based on the rational polygon and known as the Regular Polytope and specifically designed to overcome floating point and other errors induced by the finite nature of the computerised environment. A convex regular polytope (a figure with a high degree of symmetry) can be constructed as the intersection of a number of half-spaces each defined by a plane in 3D. These convex polytopes can then be combined to form a general regular polytope (i.e. a regular polytope is the union of a finite sets of convex polytopes). This can be used to represent 3D features. The author proves that a set of these regular polytopes forms the basis for a topological space. If H represents a half-space and **H** is the set of all half-spaces, a convex polytope C is defined as the intersection of any finite number of half-spaces. A regular polytope O is then defined as the union of a finite set of convex polytopes.

$C = \{H_i : H_i \in \mathbf{H}, i = 1 \ldots n\}$ **Convex Polytope in terms of Half-Spaces**

$O = \{ C_i : C_i \in \mathbf{C}, i = 1, m \}$ **Regular Polytope in terms of Convex Polytopes**

These definitions can then be used to underpin the definition of topological relationships. For example, for intersection of convex polytopes: $C1 = \{H_{i1} : H_{i1} \in \mathbf{H}, i = 1 \ldots n\}$, $C2 = \{H_{i2} : H_{i2} \in \mathbf{H}, i = 1 \ldots m\}$, the intersection (which is also a convex polytope) is defined as follows:

$C1 \cap C2 = : H_{i1} \in \mathbf{H}, i = 1 \ldots n \} U \{H_{i2} : H_{i2} \in \mathbf{H}, i = 1 \ldots m \}$ **Intersection of Convex Polytopes**

Similarly, intersection of regular polytopes is defined as:

$For \quad O_i = \{C_{i1} : C_{i1}, i = 1, n\}, O_2 = \{C_{i2} : C_{i2}, i = 1, m\}$

$Let \quad O_1 \cap O_2 = \{C_{i1} \cap C_{j2}, i = 1, n, j = 1, m\}$ **Intersection of Regular Polytopes**

As the intersection of two convex polytopes is a polytope then this intersection is also a polytope. Other operations defined on regular polytopes include union, containment and inverse (defining the exterior).

Other Frameworks

Zlatanova *et al.* (2004) identify two further frameworks in the context of 3D spatial objects – one proposed by Shi and Guo (2002), developed specifically for uncertain objects, and the second proposed by Liu and Shi (2003) who redefine topological relationships between two objects, identifying the number of topological relation between two infinites and approximating this to a series of matrices.

Appendix 3 – Topological Data Structures

The following table summarises the topological data structures reviewed.

Name	*Author(s)*	*Type*	*Description and Comments*
3DFDS (Three-dimensional Formal Data Structure)	Molenaar (1990) Relational implementation by Rikkers *et al.* (1994). Also implemented by Tempfli (1998) and Gruen and Wang (1998)	B-Rep	Based on Formal Data Structure Topological elements are Node, Edge, Arc and Face Spatial objects are Point, Line, Surface and Volume Arcs are the boundary of Faces and are oriented to avoid ambiguity, and a Face can be part of two bodies at most Can model cavities and surfaces Cannot represent irregular surfaces such as those found in earth sciences
	Tse and Gold (2003)	B-Rep	Triangulation (simplex) TIN based model OR network based approach The model defines operators that can be applied to TINS to overcome issues with 2.5D TINS with bridges TINS are modelled as Quad-Edged structures so that the Euler Operators can be applied for model validation Does not assume a complete partitioning of the 3D surface. Good extension of the TIN model which is familiar to SIS users Triangulations need to be traversed to make up geometries and identify adjacencies. This model is an extension of 2.5D situation
TEN (Triangulated Network)	Pilouk (1996)	B-Rep	Simplex based, tetrahedron based, suitable for modelling objects with indiscernible boundaries. A 3D body is a contiguous set of tetrahedra that is a subset of the TEN. A surface is a contiguous set of triangles that are Faces of the tetrahedra. A line is a contiguous set of Edges of the tetrahedra and triangles. A point object must be a Vertex of at least one tetrahedron. Can represent the internal structure of objects and is particularly suitable for geological modelling (Gong *et al.* 2004). Does not handle containment exceptions. Self-overlapping and self-intersecting objects are not allowed.
SSM (Simplified Spatial Model)	Zlatanova (2000)	B-Rep	Designed for urban modelling and support for visualisation over the internet Based on 3DFDS, but does not model the Edge element. The structure can also represent containment exceptions including Node in Face. As the Edge object does not exist, it stores fewer explicit relationships than 3DFDS which means quicker visualisation.

Name	*Author(s)*	*Type*	*Description and Comments*
VPF+ (Vector Product Format)	Ladner *et al.* (2001)	B-Rep	Wing-Edge based, with five primitives identified – entity Node (isolated points), connected Node (end points for Edges), Face (a Facet of a 3D object such as a wall), EFace (describes the use of a Face by and Edge). Supports non-manifold objects having more than two Faces adjacent to an Edge and also containment exceptions. Also allows modelling of curved surfaces through parametric equations Faces also have orientation to support visualisation requirements.
SOMAS (Solid Object Management System)	Pfund (2001)	B-Rep	Structure based on Wing-Edge approach to modelling Boundary-Representation data. Primitives of vertex, Edge, Face and Body employed. Total of 12 tables modelled, if it is assumed that object classes are merged. Coordinate information stored on the Node and information built up from this. No containment exceptions modelled.
Cell Tuple	Pigot (1995)	B-Rep	The cell tuple model was developed with a primary focus on structure population and maintenance, but topological relationships can be derived through the examination of shared cells. Uses generalised regular cell complexes are used to define objects, where cells are 0,1,2 or 3 dimensional. Cells are generalised from n-disks into Euclidean n-manifolds with one or more boundary cycles ($n \leq 3$).
UDM (Urban Data Model)	Coors (2003)	B-Rep	Like SSM, Edges are not explicitly stored (but this was developed independently) This has advantages in terms of storage space required, but Edges also need to be inferred from the Nodes when they are required Planar convex Faces required – the implementation restricts these to triangles, in contrast to SSM Cannot support holes or tunnels, as it is difficult to model these if only Nodes are used Proposes a dominance value for query results to make sure the important objects are returned from the query first
3DGT	Zeitouni *et al.* (1995)	B-Rep	Devised for urban modelling, this model supports directional adjacency, allowing queries to return values such as object A is above object B. It uses TIN surfaces to represent horizontal surfaces, with vertical planar Faces to define vertical Edges of buildings and other urban objects Internal Volume not subdivided into tetrahedrons or other structures
GTP (Generalised Tri-Prism)	Lixin and Wenzhong (2003)	B-Rep	A tri-prism has triangular top and bottom Faces Designed for subsurface geology and engineering excavation problems and focussed on processing of drill-hole data. Model has horizontal triangulated surfaces (as for 3DGT) but in addition subdivides Volume into prisms by using vertical Faces to split the Volume Model is based directly on data sampling, and can also be applied in 2.5D Horizontal planar surfaces not supported

Name	*Author(s)*	*Type*	*Description and Comments*
QTPV (Quasi-Tri-Prism Volume)	Gong *et al.* (2004)	B-Rep	Geological exploration focussed, with boreholes forming the Edge of the QTPV A QTPV is similar to a quasi-tri-prism as described above, but does not have Edges parallel to each other. A special case of this is a tetrahedron, where two or more of the Nodes in the model coincide Authors state that this structure can be used for man-made structures as well as geological structures, but do not demonstrate this. Less partitioning than TEN as Edges of the QTPV are not parallel and therefore space can be partitioned more efficiently
Circular Incident Edge Lists	Levy *et al.* (2001)	B-Rep	Aimed at visualisation of data from an unstructured grid or mesh by generating a series of isosurfaces. Suitable for dynamic fluid modelling applications Approach can also be adapted to model topology of polyhedra Half-Edge model used to model the Edges
	Wei *et al.* (1998)	B-Rep	Tetrahedron based structure, very similar to TEN
Poincare TEN	Penninga *et al.* (2006)	B-Rep	Tetrahedron structure, based on TEN but does not explicitly store the entire set of primitives. Instead, uses Poincare Simplicial homology (which states that the boundary of a simplex S_n is defined as the sum of (n-1) dimension simplexes) to develop a revised structure. In this structure tetrahedra are stored (using a list of vertices). In the strictest sense, elements of lower dimension are derived using a boundary operator, greatly reducing the amount of storage required, but the actual implementation is more generic to support requirements such as small-scale mapping.
3DSIS	De La Losa and Cervelle (1999)	B-Rep, Object-Oriented	Object oriented model which requires an iterative approach to define objects and adjacencies from the model Cannot represent the internal structure of objects (Gong *et al.* (2004))
OO3D	Shi *et al.* (2003)	B-Rep, Object-Oriented	TIN and Octree based model, using a tetrahedron based approach Model is object oriented, defined based on simplices, with 3D objects made up of an aggregation of several surfaces. Models both standard geometry and networks
OOVDM (Object-Oriented Vector Data Model)	Hou and Wu (2002)	B-Rep, Object-Oriented	Fundamental primitive is a point, with associated coordinate information. This is then aggregated into two linear object types – arc and 'characteristic line'. These in turn are aggregated to form simple, complex and composite Volumes. Topological relationships between all these levels of objects are supported, to support requirements of geological analysis. Triangulation approach taken to object construction.

Name	*Author(s)*	*Type*	*Description and Comments*
NRS (Node Relation Structure)	Lee (2001)	Graph	Model focuses only on adjacency and connectivity Uses Poincare duality to model 3D objects as Nodes – this process conserves topological properties Model particularly useful for navigation within buildings and for shortest path queries Two graphs generated – one for adjacency and one for connectivity, as adjacent rooms are not necessarily connected Author suggests matrix as a storage model for the graphs
Topological Network Model	Takino (2000)	Graph	Aims to improve navigation, information provision, internet data provision and data analysis for urban 3D data. The graph model aims to allow for different levels of detail and for management of large data Volumes. All buildings having access to a particular road are linked, and also linked to the road. Network approach allows shortest path calculation. Model designed to support automated navigation through a 3D virtual environment.
Dual Structure	Ramos (2002)	Hybrid	Dual structure, with Node, Edge, Face and Volume primitives but also incorporating a network based approach. Approach is based on the principle that a model designed for binary relationship identification is not suitable for network applications and vice versa. B-Rep is used for the binary queries, and Node/Edge used for the networking queries. Identifies a series of requirements for the structure Structure not designed to model the inside of buildings Faces are built directly from Nodes and do not include Edges

Table 89 - Topological Data Structures

Appendix 4 – The Test Dataset

The following paragraphs outline the steps required to convert the Zlatanova (2000) dataset from a series of paper-based diagrams to a replicated 1.08 million-object topologically structured dataset. The semi-automatic process used to capture and structure the data is first described, followed by an outline of the validation methods implemented to ensure that topological relationships had indeed been created correctly. The dataset to populate STS was created (264 objects) and replicated (to 1.08 Million objects) first. This was then used as a basis for the 3DFDS and As-Required datasets.

Initial Creation Process - STS

The identification of topological relationships and consequent population of a topological data structure involves the splitting of objects into associated topological primitives, identifying and storing any shared primitives and containment relationships. As a prerequisite to structure population, a clear definition of the primitives themselves must be provided – for example, is an Edge single or multi-segmented? Are all Faces triangles? This information is required in addition to the data structure described above as various approaches to structure population can be identified. A number of issues are relevant here, including the selection of a simplex or non-simplex primitive, the selection of a set of rules to govern the splitting of line, surface and body objects into component primitives and the identification of a Face type – planar or non-planar.

As discussed in Arens *et al.* (2005), the use of triangular and tetrahedral primitives has the advantage that structures are easily validated. They can also be used for 2.5D representation (Penninga 2005) support visualisation and have well known topological relationships (Penninga *et al.* 2006), which is particularly useful where 3D data is not available. However, these benefits are outweighed by the large number of primitives required to model each 3D object. Arens *et al.* (2003) also propose polyhedra, polyhedra with patches and CAD objects as options for 3D modelling. However, given that planar faces are a requirement for 3DFDS, polyhedra will be selected for implementation here, although STS can handle all of these object types provided that the underlying database can handle the corresponding primitives (if these are to be modelled).

The number of primitives required for relationship identification in 3DFDS is considerably larger than that for STS, as the latter allows structures including multi-planar Faces and multi-end Edges. Additionally, 3DFDS places more restrictions on the type of primitives selected. To ensure a fair performance comparison test, and given the unavailability of an automated topological engine to be utilised for structure population, the more restrictive 3DFDS primitive

types were utilised for structure population in all cases. The rules for the creation of these primitives from parent object geometries are summarised in Table 90 below.

Rule	*Description*
Primitive Types	Node, Edge, Face and Volume primitives are created
Nodes	All coordinate tuples in the parent object are represented as Node primitives.
Edges	Edge primitives join two Node primitives directly and consist of single segment linear geometries.
Faces	Single polygon, planar primitives. Due to the manual creation process using SQL scripts (described below), and to the creation of Faces parallel to the x, y and z axes scribed below) Face planarity is ensured by the selection of appropriate coordinate values during initial object creation. This approach was selected to simplify data creation and could be used as it is the topological relationships that are of interest rather than the geometrical configuration of the features. Note that this situation would not hold for real-world data, where validation algorithms would be required.
Node/Edge Intersection	Causes the Edge primitive to be split into two new Edges.
Edge/Face Intersection	An Edge primitive intersecting with a Face primitive does NOT cause the Face to be split unless the Edge completely crosses the Face. If the Face is not split, the Edge is marked as a containment exception in the Face. The geometrical representation of the Face primitive will be created with an appropriate hole or indentation as required.
Face/Face Intersection	A Face primitive intersecting with a Face primitive does NOT cause the Face to be split unless the Face completely crosses the Face. If the Face is not split, the second Face is marked as a containment exception in the Face.
Face/Volume Intersection	A Face primitive does NOT split Volume primitive, unless the Face itself forms the boundary of an enclosed Volume contained within the outer Volume.

Table 90 - Rules for Primitive Creation

The creation of topology engine algorithms is complex and beyond the scope of this thesis. To circumvent this complexity, initial attempts to populate STS involved writing SQL scripts to create the geometry and the populate required records in the TOPO_ tables, as well as those in the NODE, EDGE, FACE and VOLUME tables. As this task proved time-consuming and error prone a semi-automated method was developed to support the process.

Implementing 3D Objects in Oracle Spatial

Oracle Spatial provides an SDO_GEOMETRY object for the storage of spatial data which has a structure as shown in Table 91 below.

Key	*Description*	*Sample Values*
SDO_GTYPE	Indicates the type of geometry. *d* is the dimension of the embedding space. *l* is utilized for linear referencing, and is set to 0 for all other objects.	*dl*01 – point *dl*02 – line or curve *dl*03 – polygon, no holes *dl*04 – collection of all other types
SDO_SRID	Spatial Reference Identifier	81989 for British National Grid
SDO_POINT	A point object, used instead of SDO_ELEM_INFO and SDO_ORDINATE_ARRAY to define X, Y and Z coordinates for point data.	Usually set to NULL, NULL for non-point datasets.
SDO_ELEM_INFO	Describes the individual elements in the geometry – in particular where the geometry is made up of a collection. The SDO_ELEM_INFO is made up of a triplet of values for each element – the STARTING OFFSET (the number of the first coordinate in the array that describes the element), the ELEMENT TYPE (simple or compound, exterior or interior polygon ring), the INTERPRETATION – which defines the subsequent elements for compound objects.	ETYPE is 1 for points, 2 for lines, 1003 for external polygons and 2003 for internal polygons.
SDO_ORDINATES	A list of the coordinates making up the object. These are used in conjunction with the SDO_ELEM_INFO to define the individual components of the object.	

Table 91 - The SDO_GEOMETRY Object

There are currently no standardized methods for storing a 3D Body in an SDO_GEOMETRY object, although these will be included in the next release of Oracle (Oracle 2007). It is difficult to distinguish between a multi-Face surface and a body, as both make use of the standard collection GTYPE of 3004. Arens (2003) proposes a new G-TYPE VALUE to handle body objects, which is suggested to be 3008. However, using custom data types prevents the standard Oracle spatial indexes from being compiled against the dataset. Using existing GTYPE values overcomes this issue and a number of researchers have implemented such structures. Arens *et al.* (2005) make use of a 3D line object, also including Nodes in the structure to be able to construct the required Faces. This approach does not aim to support topological relationships between objects, although topology within the object is preserved. Breunig and Zlatanova (2005) propose using a multi-polygon collection type of 3004 in conjunction with a BODY table – which thus distinguishes the body object from the surface. They also list an alternate option – the use of a relational approach, splitting the body object into a list of polygon objects linked through a join relationship.

Although the SDO_GTYPE of 3004 can be utilised to represent the majority of the 3D GEOMETRY objects in the dataset, modifications to the standard SDO_GEOMETRY structure

were required to resolve two issues specific to the semi-automated population of the topological structure. Firstly, information was required to allow the semi-automated process to distinguish between multi-Face surfaces and polyhedra. Secondly, information regarding the containment relationships to be stored as additional links in the TOPO_ tables was also included artificially in the SDO_GEOMETRY object, as calculating these would involve the development of a topological engine. The changes described below were made purely to support the population of the topological dataset, evolving on a case-by-case basis as the dataset was created, rather than with any standardisation of a 3D structure in mind. This resulted in a rather disconnected series of modifications to the standard SDO_GEOMETRY structure, each one designed for a specific purpose and for one-off usage.

To distinguish between multi-Face surfaces and body objects, a COLLECTION_TYPE field was added to the geometry table[20] taking a similar approach to that described by Breunig and Zlatanova (2005). This stored information relating to whether the geometry is a simple multi-part surface, a self-intersecting multi-part surface, a simple Volume, a multi-part Volume with no cavities but with internal objects, a multi-part Volume with cavities or a multi-type object. COLLECTION_TYPE codes were numbered from 1 to 6. Modifications are shown in Table 92 below.

[20] Note that this COLLECTION_TYPE information was added as an extra field rather than to the SDO_GEOMETRY data type as the aim of the process was to facilitate the creation of a single set of test data. No formal attempt was made to design systematic modifications for SDO_GEOMETRY for use beyond this purpose.

Collection Type	*SDO_GTYPE*	*SDO_ETYPE*	*Description and Semi-Automatic Processing*
N/A	3001	N/A	Simple Point object – process by creating Node
N/A	3002	N/A	Simple Line object – create the corresponding Nodes, create the Edge (or Edges) and update the IS_BOUNDARY flags for the end Nodes as these are boundary objects for the Edges
1	3004		Multi-part Surface not self-intersecting. Process as for simple surfaces, flagging associated Edges and Nodes as boundary, unless they are repeated in the collection (i.e. Edges shared between two Faces of the surface). All Faces are set to IS_BOUNDARY = FALSE.
2	3004		Self-intersecting multi-part surface – process manually
3	3004		Simple Body – process the Volume primitive, creating Faces, Edges and Nodes. All primitives set to IS_BOUNDARY = TRUE except for the Volume primitive.
4	3004		Multi-part body – i.e. a body with something contained inside it – but with no internal cavity – first process the Volume primitives, then process as for a multi-part Surface. If a Face is repeated in the SDO_GEOMETRY ordinate list, then this is contained within the Body and should be flagged as IS_BOUNDARY = false. All other Faces are flagged as IS_BOUNDARY = true. Note that for body-in-body situations, the association of the inner Volume to the outer one is created manually once the Semi-Automatic population code has been run.
5	3004		Multi part body with internal cavity – not used
6	3004		Multi-part object e.g. surface with lines – not used
4	3004	2003, 2	Isolated line (2) or hole (2003) contained in the Face primitive that has just been listed in the SDO_GEOMETRY ordinate list. Set primitive to IS_BOUNDARY = true.
4	3004	1	Isolated Node on the Face that has just been listed in the ordinate list. Set Node primitive to IS_BOUNDARY = true.
4	3004	9999	Exceptions in the Volume primitive that has just been listed in the ordinate list. These relate to Edges and their end-Nodes. Set primitives to IS_BOUNDARY = false.

Table 92 - Modifications to SDO_GEOMETRY to support STS Structure Population

Two approaches to the modelling of containment relationships were also implemented. The repetition of a Face in an SDO_GEOMETRY representing a body object indicated that the Face was contained in the body. For Edges, E-TYPE 9999 was used to identify the Edge in Body relationship.

Figure 108 below shows the SQL used to create a body object with an artificially associated internal Faces (Feature ID 73, representing the Covers relationship R435). Note that the inner Faces are repeated in the SDO_GEOMETRY structure to represent the fact that they are actually contained within the object. Three outer surfaces of the body are also artificially split

into multiple Faces (this task would normally be carried out by the topological engine). The R435 relationship is shown in Figure 109.

```
INSERT INTO FEATURE(FEATURE_ID,COLLECTION_TYPE,GEOMETRY)
VALUES /*THE OUTER BOX*/
(73,4,MDSYS.SDO_GEOMETRY(3004,NULL,NULL,
MDSYS.SDO_ELEM_INFO_ARRAY(
1,1003,1,22,1003,1,  -- FRONT LARGE AND FRONT SMALL FACES
37,1003,1,58,1003,1, -- BOTTOM LARGE AND SMALL FACES
73,1003,1,94,1003,1, -- RH SMALL AND LARGE FACES
109,1003,1, -- TOP
124,1003,1, -- LH SIDE
139,1003,1, -- BACK
154,1003,1,169,1003,1, -- TOP INNER FACE, TOP INNER FACE REPEATED
184,1003,1,199,1003,1, -- BACK INNER FACE, BACK INNER FACE REPEATED
214,1003,1,229,1003,1), -- LEFT INNER FACE, LEFT INNER FACE REPEATED
MDSYS.SDO_ORDINATE_ARRAY(
… )))
/
INSERT INTO GEOMETRY(GEOMETRY_ID,RELATIONSHIP_ID, COMPONENT,COLLECTION_TYPE,GEOMETRY)
VALUES /* THE INNER BOX*/
(74,435,'B',3,MDSYS.SDO_GEOMETRY(3004,81989,NULL,MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1,
16,1003,1,31,1003,1,46,1003,1,61,1003,1,76,1003,1),
MDSYS.SDO_ORDINATE_ARRAY(
… )))
/
```

Figure 108 - SQL to Capture Covers Relationship R435

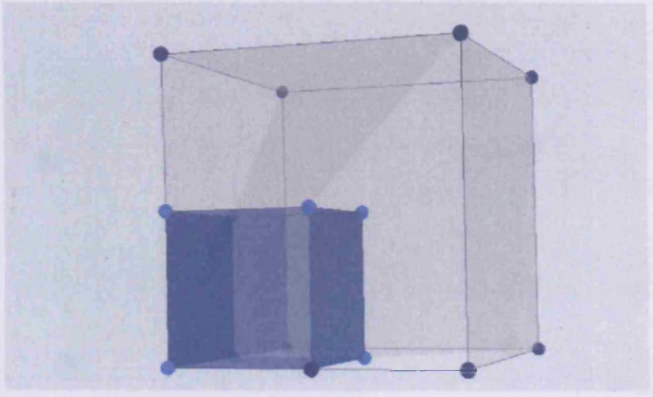


Figure 109 – Relationship R435

Figure 110 represents the SQL used to create an SDO_GEOMETRY object for a Body intersecting a Line, where part of the Line is contained within the Body. The primitives containing the line are associated with the Body geometry to allow the semi-automatic structure creation process to detect the containment relationship. A separate SDO_GEOMETRY object is also created for the Line itself to show that it is a separate object rather than part of the Body (FEATURE_ID 83). R191 is shown in Figure 111.

```
INSERT INTO FEATURE(FEATURE_ID, COLLECTION_TYPE,GEOMETRY)
VALUES
(83,4,MDSYS.SDO_GEOMETRY(3004,NULL,NULL,
MDSYS.SDO_ELEM_INFO_ARRAY
(1,1003,1,16,1003,1,31,1003,1,46,1003,1,61,1003,1,76,1003,1,  -- THE BODY ITSELF.   TOP
FACE IS THE LAST ONE LISTED
91,1,1, -- ISOLATED NODE ON THE TOP FACE
94,1,1, -- SECOND ISOLATED NODE ON THE TOP FACE
97,9999,1, -- INERNAL EDGE 1
103,9999,1, -- INTERNAL EDGE 2
109,9999,1), - INTERNAL EDGE 3
MDSYS.SDO_ORDINATE_ARRAY(
…
)))
/
INSERT INTO FEATURE(FEATURE_ID,RELATIONSHIP_ID, COMPONENT,COLLECTION_TYPE,GEOMETRY)
VALUES
(84,191,'A',1,MDSYS.SDO_GEOMETRY(3002,81989,NULL,MDSYS.SDO_ELEM_INFO_ARRAY
(1,2,1,7,2,1,13,2,1,19,2,1,25,2,1,31,2,1),  -- THE LINE FEATURE
MDSYS.SDO_ORDINATE_ARRAY(
…
)))
/
```

Figure 110 - SQL Script for Relationship R191

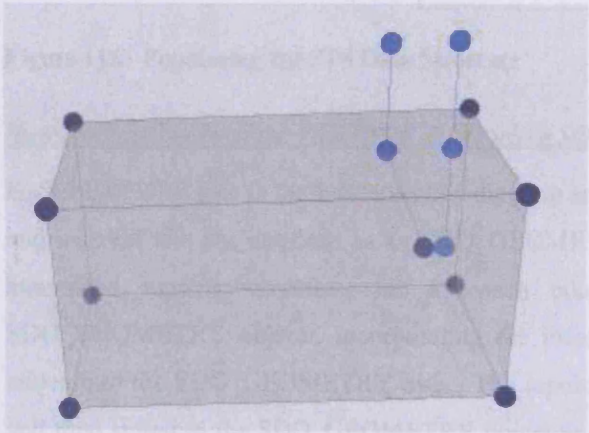


Figure 111 - Relationship R191 between a Body and a Line

Semi-Automated Data Structure Population

An overview of the structure population process is given in Figure 50 below.

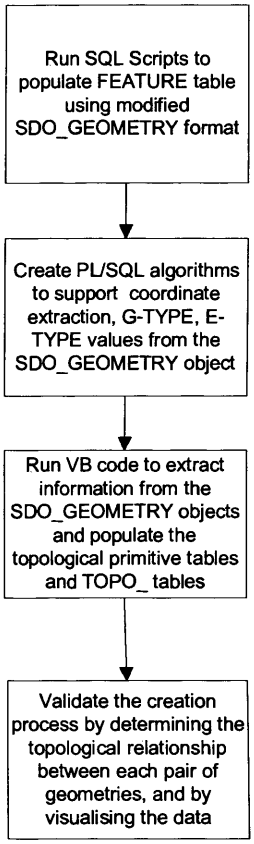


Figure 112 - Populating the STS Data Structure

Step 1 – Population of the FEATURE Table using SQL Scripts

Each FEATURE pair in the 9-Intersection diagram set was manually captured using SQL scripts and inserted into the database as an SDO_GEOMETRTY object. However, to support semi-automated topology creation, the approach taken involved the scripting of modified SDO_GEOMETRY objects, incorporating the information required to populate the TOPO_ tables into the SDO_GEOMETRY data. The topological relationships were defined manually and then stored in the SDO_GEOMETRY object as described above. To facilitate the manual creation of the SQL scripts for the objects and to ensure planar surfaces, all SDO_GEOMETRY coordinates were first defined around the origin (0,0,0) with maximum values for X, Y and Z all less than 10. All data was initially described using integer coordinates, with axes-parallel edges.

Step 2 – Extraction of SDO_GEOMETRY information using PL/SQL Routines

Following on from the creation of the modified SDO_GEOMETRY objects described above, routines were devised to extract of coordinate and object-description information from the records in the database. These were required to underpin both the shifting of the data from the

original object coordinates around (0,0,0) to ensure that geometry did not overlap unless designed to do so, and the replication of each object. As PL/SQL provides an optimal database programming language where Oracle data types and intensive database queries are required, this was used to create a number of packages to support these processes.

The code developed includes a function to list all the SDO_ORDINATES in an SDO_GEOMETRY object, another to extract the individual components of the G-TYPE and E-TYPE information. Each function returned a PL/SQL table, allowing the function to be incorporated into SQL standard queries. These functions were also used to support the data replication process described below.

Step 3 – Population of the Topology Structure using Visual Basic Routines

The first part of the structure population process involved shifting the geometry from (0,0,0) along the X axis to ensure that objects did not occupy the same space unless they formed part of a relationship pair. Once this was completed, entries were created in the TOPO_PART_TABLE (one for each object) and NODE and EDGE primitives created by using the PL/SQL procedures described in Step 2 to extract coordinate information from the FEATURE objects. Each line segment was created as a separate Edge object, and the end Nodes and intermediate Nodes were also stored as separate primitives. This process was applied to all objects as the minimum dimension of the dataset is 1. TOPO_NODE and TOPO_EDGE associations with the FEATURE (via the TOPO_PART_TABLE) were created at each stage of this process.

The combination of COLLECTION_TYPE and E_TYPE values were used to identify and define FACE and VOLUME primitives, and TOPO_FACE and TOPO_VOLUME associations created. The entire dataset was then re-parsed to validate IS_BOUNDARY settings, utilizing the E_TYPE and repeated Faces to determine containment relationships and hence set IS_BOUNDARY as required. Object dimension was also examined to support this process – for example, Faces forming part of a Body object are always boundary unless a containment relationship has been identified – i.e. unless the coordinates for the Face appear twice.

Finally, a SQL script update process was utilized to manually correct a number of issues outstanding following semi-automatic structure population. These related primarily to the absence of a routine to identify and reuse existing VOLUME primitives. A number of minor data errors, resulting from the incorrect creation of the source object, were also corrected.

Validation – STS

A three-part validation process was used following structure population. Testing the correct creation of the dataset using the 9-Intersection queries provides the first element of the validation process. Visual validation of the primitives, particularly with regard to non-shared elements of each relationship pair, is also fundamental to improving the quality of the dataset as a whole. The dataset has primarily been created for topological purposes, and thus the SDO_GEOMETRY objects associated with each primitive are not utilised for data structure comparison. However, if the dataset is to be more widely disseminated and used for other purposes these objects may be more important. A final stage of testing involved querying the data structure using standard SQL to ensure that, for example, all Edges were associated with two Nodes.

Testing the Data using PL/SQL Procedures

A generic test routine and associated test harnesses determined the 9-Intersection relationships for each pair of geometries in the dataset (implementing the algorithms described in Chapter 7), and compared the results obtained against the R-Value in the corresponding diagrams in Zlatanova (2000). A table to store test results was created, and for each pair of objects, the 9-Intersection value in calculated and stored alongside the expected R-values for each relationship. Querying the tables identified object pairs reporting incorrect topological relationships. This was due to errors in the original scripts, which were created by manually typing the required SQL (for example, the use of incorrect coordinates on one of a pair of features, resulting in no shared primitives being identified) and to errors in the Visual Basic code for the semi-automated process (for example, the process did not detect shared VOLUME primitives). A total of 63 errors were corrected (out of a total of 7899 TOPO_ and primitive records created) using SQL INSERT and UPDATE statements to add missing primitives and correct Feature and Primitive geometry. The primitive creation and TOPO_ population process was repeated until all issues were resolved. The number of original errors highlights the issues when creating geometries manually and confirms the decision to semi-automate the process.

Visualising the Data

A 3D visualization tool was developed provide independent confirmation that the data had been captured correctly. This tool was used to visualize the Node, Edge and Face primitives associated with an object. Java was chosen as the development platform for this tool, with the Java3D toolkit providing the required 3D functionality. An overview of the visualization process is given in Figure 113 below. The visualization tool was developed by Roy (2005) and the work to zoom and scale individual objects to the centre of the screen was carried out by Tse (2006).

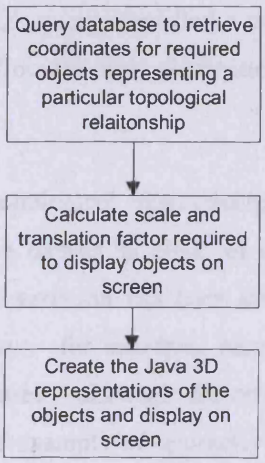
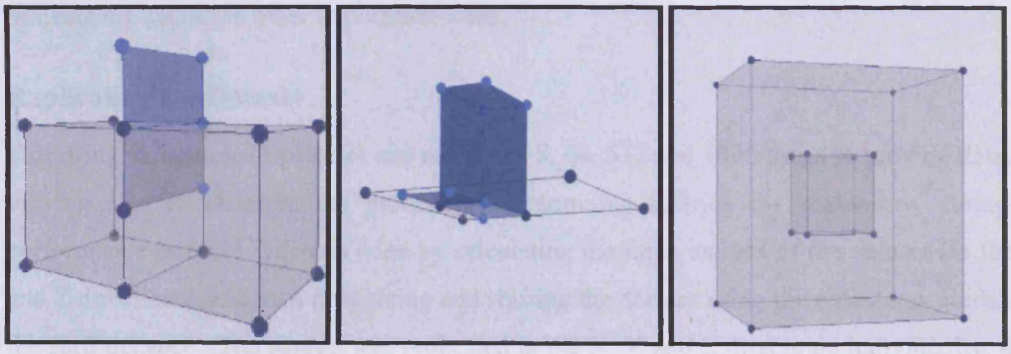


Figure 113 - Visualising STS Data

A query is retrieves the pair of Oracle SDO_GEOMETRY objects from the database, using FEATURE_ID as the key. The SDO_ORDINATES of the objects are then available directly from this feature class and can be used to determine the centroid and Minimum Bounding Volume of the objects. This information supports the identification of appropriate translation and scale factor to display the object, based on the work carried out by Tse (2006) – these are required as Java3D's default display area is centred around the origin (0,0,0) with extents of ±0.5 units in either direction. A transformation is created using the translation and scale factors and this is applied to the dataset to reset the display centre and scale to values appropriate for the objects in question.

Figure 114 below shows a number of examples of the result of the visualization process.



(a) (b) (c)

Figure 114 - (a) Body/Surface R511 (b) Surface/Surface R221/R183 (c) Body/Body R179

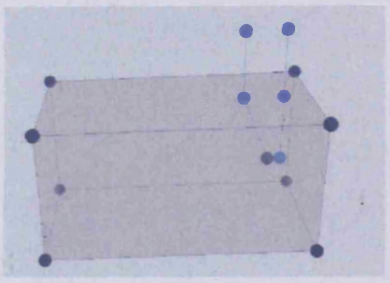
Testing the Data through SQL Queries

SQL queries were utilized to ensure that the correct number of primitives had been created for each object, and in particular that the containment relationships were correctly modelled. The

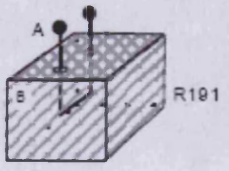
combination of visualization and SQL query identified a number of minor errors with the data, which were corrected following on from the semi-automatic structure population process.

Results of the Testing Process

The test process validated all topological relationships using the PL/SQL algorithms. Visualisation also revealed that the dataset is close in appearance to that diagrammed by Zlatanova (2000). However, some variation has been introduced. This primarily relates to small modifications to the geometry – for example, contained lines that are vertical in the diagrams may not be so in the dataset – although the actual primitives associated with each object will be as diagrammed. An example of geometry variations is shown in Figure 115 below for relationship R191 between a body and a line, where Figure 115(a) shows the resulting geometry and Figure 115(b) the original diagram from Zlatanova (2000).



(a)



(b)

Figure 115 - Geometry variation for R191 Body/Line

The validation process focuses on topological relationships. Further errors may be revealed by utilising the dataset in other application areas.

Replicating the Dataset

Following validation, the dataset was replicated 8, 64, 512 and 4096 times to provide datasets of varying size to underpin the process of determining metrics for scalability, storage and performance testing. This was done by calculating the three extents of the dataset (in the X, Y and Z directions), and then duplicating and shifting the dataset using the extents as guidance for the shift distance. The dataset was replicated in the X, Y and Z directions individually, then in XY, YZ and XY, and finally in XYZ (with each replica coded 1-7). Each replication therefore resulted in a dataset eight times the previous size.

The replication process utilized the PL/SQL coordinate geometry, G_TYPE and E_TYPE routines described above to calculate the new position for each new geometry object in the main

table, and also for each new SDO_GEOMETRY representing the Node, Edge and Face primitives. In summary, replication steps were therefore as shown in Figure 116.

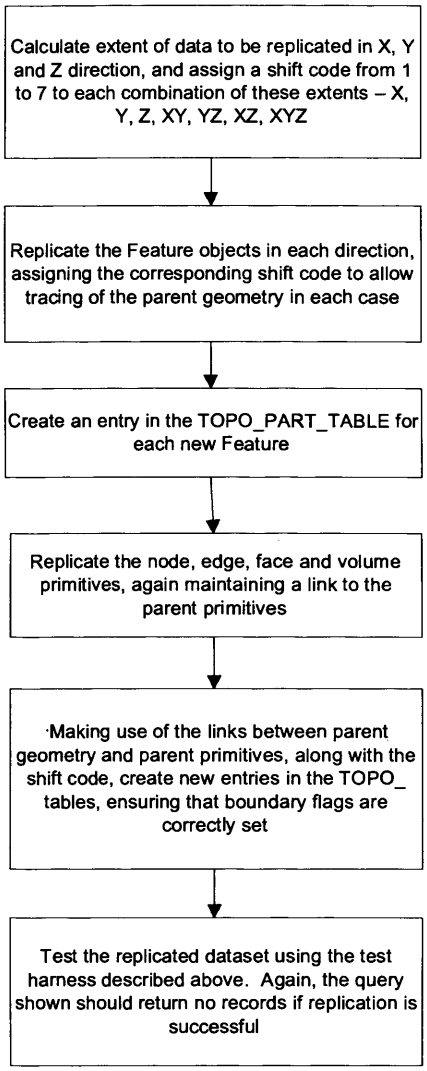


Figure 116 - Flow Process for Dataset Replication

To support the replication process described, OLD_PRIMITIVE_ID and SHIFT_CODE fields were added to the FEATURE, NODE, EDGE, FACE and VOLUME tables. The shift code identifies the replication code from 1 to 7. The OLD_PRIMITIVE_ID creates the link to the original geometry or primitive ID from which the record was replicated. These fields were indexed to improve the performance of the data replication processes, and allowed all the primitives and the Feature objects to be created as a batch process prior to the population of the TOPO_ tables, as attempting to populate the TOPO_ tables for as each primitive was processed resulted in the violation of foreign key constraints in the database. An example of the replication algorithm is given in Figure 117 for the EDGE table, replicating to eight times the original number of records.

ALGORITHM TO REPLICATE EDGE DATA

- Identify the minimum and maximum X, Y and Z bounding coordinates for the dataset.
- Identify values for Δx, Δy and Δz, the extents of the dataset.
- Take a copy the existing EDGE primitives (assuming that the original dataset is to be kept intact).
- Select all the EDGE primitives
- For each primitive in the selection:
    - Determine the EDGE_ID of the primitive (for example 1021)
    - Retrieve the SDO_GEOMETRY object representing the primitive
    - Replicate 7 times
        - *Replica 1* – add Δx to the x coordinates, create a new SDO_GEOMETRY object and insert this into the EDGE table with a new EDGE_ID. Set OLD_EDGE_ID to be the EDGE_ID (1021) of the primitive being replicated. Set SHIFT_CODE to 1, to indicate that the replica involved only a Δx shift in coordinates.
        - *Replica 2* – repeat the above, but for Δy shift, with the same OLD_EDGE_ID (1021) and SHIFT_CODE = 2
        - *Replica 3* – repeat the above, but for Δz shift, with the same OLD_EDGE_ID and SHIFT_CODE = 3
        - *Replica 4* – repeat the above but for both Δx and Δy shifts with the same OLD_EDGE_ID and SHIFT_CODE = 4
        - *Replica 5* – repeat the above but for Δx and Δz shifts, with the same OLD_EDGE_ID and SHIFT_CODE = 5
        - *Replica 6* – repeat the above but for Δy and Δz shifts, with the same OLD_EDGE_ID and SHIFT_CODE = 6
        - *Replica 7* – repeat the above but for Δx, Δy and Δz shifts, with the same OLD_EDGE_ID and SHIFT_CODE = 7

Figure 117 - Algorithm to replicate Edge Data

Understanding how data is stored on disk is also important when creating a large dataset. Conceptually, data is stored in blocks on the disk, where a block is the amount of data read into memory at one time. As performance testing is one of the potential uses of this dataset, it was important to ensure that that the data created by replication represents real-world data as closely as possible. In particular, two objects having a topological relationship should not be created sequentially (and thus be adjacent on disk), as this may not be representative. If this were not the case, adjacent objects would require one disk read rather than two to bring the required data into memory and thus identify the topological relationship. As disk reads are one of the most time-consuming processes when handling data (Atzeni *et al.* 1999) the single disk read may lead

to artificially rapid performance results. The replication process took account of this by first replicating odd-numbered Features and then replicating even-numbered FEATUREs.

Migration to Extended 3DFDS

3DFDS enforces the relationship hierarchy between Node, Edge, Face and Volume primitives and as described in Chapter 7, the algorithm utilised for the determination of binary topological relationships in 3DFDS utilises this hierarchy to identify, for example, any Nodes associated with an object. Migrating data from STS to 3DFDS therefore involved ensuring that all data followed this hierarchy.

Data Migration – Extended 3DFDS

Given that the STS structure was populated first, it was possible to migrate the topological datasets to the 3DFDS structure, rather then derive them from the SDO_GEOMETRY associated with the object. However, the inclusion of the EXCEPTION tables in the Extended 3DFDS structure required the specification of a clear set of rules to ensure that, as far as possible, information is not stored inconsistently in the structure. For example, Node primitives recorded in the Node-In-Volume exception table also appear in the Node primitive table. To encode the *9-Intersection Pairs* functionality against the structure, it is important to understand the situations in which this would occur.

The rules for 3DFDS structure population are summarized in Table 93 below.

Rules for 3DFDS Data Structure Population
Only the highest dimension primitive is associated with the object through the TOPO_ tables – therefore a Body will have an associated record in the TOPO_VOLUME table, but not in TOPO_FACE, TOPO_EDGE or TOPO_NODE.
If an Edge is associated with a Surface object twice (i.e. is associated with TWO of the Faces making up that Surface) then the Edge is an interior primitive of the surface.
If an Edge primitive is associated only once with a Body object, then it is an exception (i.e. contained within the body). As a body is represented by closed Faces, Edges will normally be listed twice when the Edges making up a Body are queried, once for each Face they are associated with.
If a Node primitive is only returned once by a query to identify the Nodes forming a body, then it is contained within the body.
If a Node primitive is only returned once by a query to identify the Nodes forming a Face, then it is contained on the surface of that Face.
Edges that are contained on Faces should not appear in the EDGE_FACE table linked to that Face. The EDGE_FACE table should only contain boundary Edges of Faces.
Faces primitives that are contained within Volumes appear twice in the FACE_VOLUME table, once with the Volume ID on the left and once with the Volume ID on the right.
Volume primitives contained within other Volume primitives should be referenced in the TOPO_VOLUME table, and both should be associated with the parent object. This enforces the no-overlapping-topology primitives - therefore the object is made up from the outer Volume with a cavity, plus the inner Volume (unless the object itself has a cavity)
Similarly, for surface objects with inner Faces, there should be 2 references in the TOPO_FACE table - the outer Face with a hole and the inner Face.
If the only Edge linking to a Node-in-Face exception is also an exception in the same Face (i.e. the Node exceptions are only present as they form the end points of the Edge), then there is no requirement to populate the NODE_IN_FACE exception table
If Edges are contained within a Body only due to the fact that the Face they surround is also contained within the body, there is no requirement to populate the EDGE_IN_VOLUME exception table.

Table 93 - Rules for 3DFDS Data Structure Population

As primitives created for 3DFDS are identical to those for STS, these were copied directly. An algorithm was then developed in Visual Basic to query the STS structure and transform the data to populate the 3DFDS structure. For example, Node and Edge primitives having IS_BOUNDARY = 0 and associated with a Body object were inserted into the NODE_IN_VOLUME and EDGE_IN_VOLUME exception tables. It is possible to identify a list of all the Nodes, Edges, Faces and Volumes associated with an object through STS. However, STS does not hold information with regard to the individual relationships between Node and Edge primitives, Edge and Face primitives and Face and Volume primitives. These were therefore derived by examining the SDO_GEOMETRY objects associated with each primitive, using the PL/SQL coordinate extraction routines described above.

Data Replication – Extended 3DFDS

As well as allowing population of the 3DFDS structure according to the guidelines provided by Molenaar (1992), the selection of simple primitives (single-segment Edges and the use of Nodes for each co-ordinate tuple) also facilitated the replication of data stored in this structure, as the replicated primitives are identical to those stored in STS and could thus be directly copied. The

replication process therefore involved replicating the tables identifying the relationships between these primitives (NODE_EDGE, EDGE_FACE and so forth) and also those linking the primitives to the parent objects.

A similar approach to that described for STS was followed for replication. Again, replica object pairs in were created across three dimensions and an OLD_PRIMITIVE_ID and SHIFT_CODE fields added to the Node, Edge, Face and Volume primitive tables. An overview of the replication algorithm used for the NODE_EDGE table is given in Figure 118.

ALGORITHM TO REPLICATE NODE_EDGE DATA

1. Select all the entries in the NODE_EDGE table
2. For each entry in the table (for example having NODE_ID 123 and EDGE_ID 299):
    a. Query the NODE primitive table to determine all the NODE_IDs having OLD_NODE_ID = 123. There will be 7 in total, each with a different SHIFT_CODE
    b. Query the EDGE primitive table to determine all the EDGE_IDs having OLD_EDGE_ID = 299. There will be 7 in total, each with a different SHIFT_CODE
    c. For each SHIFT_CODE, create a new entry in the NODE_EDGE table, having the NODE_ID and EDGE_ID corresponding to that SHIFT_CODE value. This creates a total of 7 new entries for each existing entry in the NODE_EDGE table.

Figure 118 - Replicating the NODE_EDGE table

Validating the Migrated Data

Given that the SDO_GEOMETRY representation of the objects and the primitives is identical to that utilised for STS, no further validation was carried out on this aspect of the 3DFDS structure.

As with the STS structure, PL/SQL algorithms developed to query the 9-Intersection relationships were initially used to validate that the dataset was correctly migrated. A generic test routine and test harness was written to validate the topological structure population process. This iterated over each object pair, and determined the R-Code for the 9-Intersection relationship for each pair of objects in the dataset (using the algorithm implementations described in Chapter 7). A table to store the results of these tests was created, and for each pair of geometries, the 9-Intersection value in calculated and stored alongside the expected R-values for each relationship.

Migration to As-Required Data Structure - SDO_GEOMETRY

As outlined above, a modified version of Oracle's SDO_GEOMETRY data type was originally utilised to facilitate the creation and population of the STS structure. This customised version includes, for example, duplicate Face polygons to represent Face containment and additional, non-standard E-TYPE values to represent topological relationships. However it is not possible to create a valid R-Tree index on a customised version of this data type. As the index is required to underpin the SDO_RELATE query utilised as a Proxy for an As-Required topological relationship identification process, a valid SDO_GEOMETRY object must first be created, omitting any primitives artificially inserted to support the STS population process.

Primitives from the populated STS structure were therefore utilised to reconstruct the correct SDO_GEOMETRY for the object.

Data Migration – As-Required Structure

Figure 119 outlines the algorithm implemented to reconstruct correct SDO_GEOMETRY objects representing the object from the STS primitives.

ALGORITHM TO MIGRATE FROM STS TO CORRECT SDO_GEOMETRY

1. Identify the dimension of the Feature from the highest primitive dimension
2. Given the Feature type:
    a. If the Feature is a Body feature, query the TOPO_FACE table to identify the Faces forming the surface of this body. Ignore any Faces having IS_BOUNDARY = 0 as these represent containment relationships
    b. If the Feature is a Surface, query the TOPO_FACE table
    c. If the Feature is a Line, query the TOPO_EDGE table
    d. If the Feature is a Point query the TOPO_NODE table
3. Query the corresponding primitive table to retrieve the SDO_GEOMETRY representation of these primitives
4. Construct an SDO_GEOMETRY object from these primitives, using ETYPE 3004 (a collection) where appropriate. For example, build the Body feature as a collection of the associated SDO_GEOMETRY representing the Face primitives. Where appropriate, examine the SDO_GEOMETRY of the primitives to determine the correct structure for the Feature – for example, to ensure that line segments are ordered correctly.
5. Insert the resulting SDO_GEOMETRY into the Feature table

Figure 119 - Algorithm to Create Corrected SDO_GEOMETRY Objects

Validating the Migrated Data

Once this process was complete, it was possible to create an R-Tree index on the data. This acted as an initial validation routine, as the R-Tree index can only be created where data is correctly structured. Similarly to the approach taken for STS and 3DFDS, a test routine was also written to determine objects returned by the SDO_RELATE query for each object pair, using a small index tolerance (0.05m). Given the small tolerance, only objects not disjoint were returned by the query. The test routine iterated over each object pair in the table, and determined the number of returned objects for the above query. The results were then automatically inserted into a test results table containing expected results. This could then be queried to determine where actual and expected results did not match.

Replicating the Migrated Data

The routine developed to generate correct SDO_GEOMETRY objects for the original dataset was reutilized for all replicated datasets, generating five Feature tables holding corrected SDO_GEOMETRY objects, ranging in size from 264 records to 1.08 million records. These Feature tables were copied at each level of replication and re-indexed using the selected index tolerance values. Six copies were made of each replicated table, to allow tests using index tolerances ranging from 0.05m through 0.5, 1.0, 5.0 and 100.0m to 500.0 m.

Sizing the Datasets

The following processes were used to determine the storage requirements of each of the replicated datasets:

Determining Table Size

Oracle provides a two-step query to determine the storage size of each table, as shown in Figure 120:

```
ANALYZE TABLE << TABLENAME>> COMPUTE STATISTICS

SELECT NUM_ROWS * AVG_ROW_LEN/1024/1024 FROM DBA_TABLES WHERE TABLE_NAME = '<<
TABLENAME>>'
```

Figure 120 - SQL Query to Determine Table Size

The first statement computes the actual statistics for the selected table, and populates a metadata table called DBA_TABLES with the results of this process. The second statement queries the results obtained, converting the answer into Megabytes of data. The above query can be run for multiple tables, with each result entered into a row in the DBA_TABLES column.

Determining Index Size

The size of a standard index is determined as follows (Figure 121):

```
ANALYZE INDEX << INDEX NAME>> VALIDATE STRUCTURE

SELECT USED_SPACE FROM INDEX_STATS WHERE NAME = '<< INDEX NAME>>'
```

Figure 121 - SQL Query to Determine Non-Spatial Index Size

In this case, it is only possible to determine these statistics for one index at a time, as the values are over written each time the ANALYZE INDEX command is run. Given this, a PL/SQL routine was created to extract information for all indexes for a specified user, as follows (Figure 122):

```
CREATE TABLE INDEX_STATS_COPY AS SELECT * FROM INDEX_STATS;

BEGIN
FOR IND_REC IN (SELECT * FROM USER_INDEXES ) LOOP
EXECUTE IMMEDIATE 'ANALYZE INDEX ' || IND_REC.INDEX_NAME ||
' VALIDATE STRUCTURE';

INSERT INTO INDEX_STATS_COPY
(SELECT *
FROM INDEX_STATS) ;
END LOOP;
END;
/
COMMIT;


SELECT USED_SPACE FROM INDEX_STATS WHERE NAME = '<< INDEX NAME>>'
```

Figure 122 – PL/SQL Query to Determine Non-Spatial Index Sizes

As the spatial R-Tree index is non-standard (i.e. not B*Tree), two separate processes are required to determine index size for the datasets created. The size of a spatial index can be determined from the size of the associated system table, which in turn stores the index information. This is known as an MD$ table. The first part of the process described below identifies the actual MD$ table associated with the index. Once this is achieved, SQL similar to that shown for table size determination can be run to determine the size of the MD$ table. This represents the size of the index. Again, a PL/SQL procedure can be used to perform this task for all spatial indexes owned by a specific schema (Figure 123).

```
SELECT SDO_INDEX_NAME, SDO_INDEX_TABLE FROM SDO_INDEX_METADATA;


BEGIN
FOR IND_REC IN (SELECT * FROM SDO_INDEX_METADATA ) LOOP
EXECUTE IMMEDIATE 'ANALYZE TABLE ' || IND_REC.SDO_INDEX_TABLE ||
' COMPUTE STATISTICS';
END LOOP;
END;
/
```

Figure 123 – SQL and PL/SQL Query to Determine Spatial Index Sizes

The resulting analysis can be queried as follows (Figure 124):

```
SELECT C.SDO_INDEX_NAME, C.TABLE_NAME, (D.NUM_ROWS * D.AVG_ROW_LEN/1024/1024) AS MB
FROM (SELECT A.SDO_INDEX_TABLE, A.SDO_INDEX_NAME, B.TABLE_NAME FROM SDO_INDEX_METADATA A
INNER JOIN USER_INDEXES B
ON A.SDO_INDEX_NAME = B.INDEX_NAME) C
INNER JOIN USER_TABLES D
ON C.SDO_INDEX_TABLE = D.TABLE_NAME
```

Figure 124 – Querying Spatial Index Size Output

The Resulting Dataset

Diagrams illustrating the resulting dataset can be found on the attached CD.

Appendix 5 – Algorithm Design

This appendix gives an overview of the logical design of the 9-Intersection component determination algorithms not described in Chapter 7. The designs are presented in the form of a flow diagram, along with a brief description of the algorithm to be implemented. Algorithms are first presented for the identification of various components of the 9-Intersection relationship between Part Objects. This is followed by an overview of algorithms relating to relationship identification for whole objects.

Boundary of Part Object A intersecting Boundary Part Object B

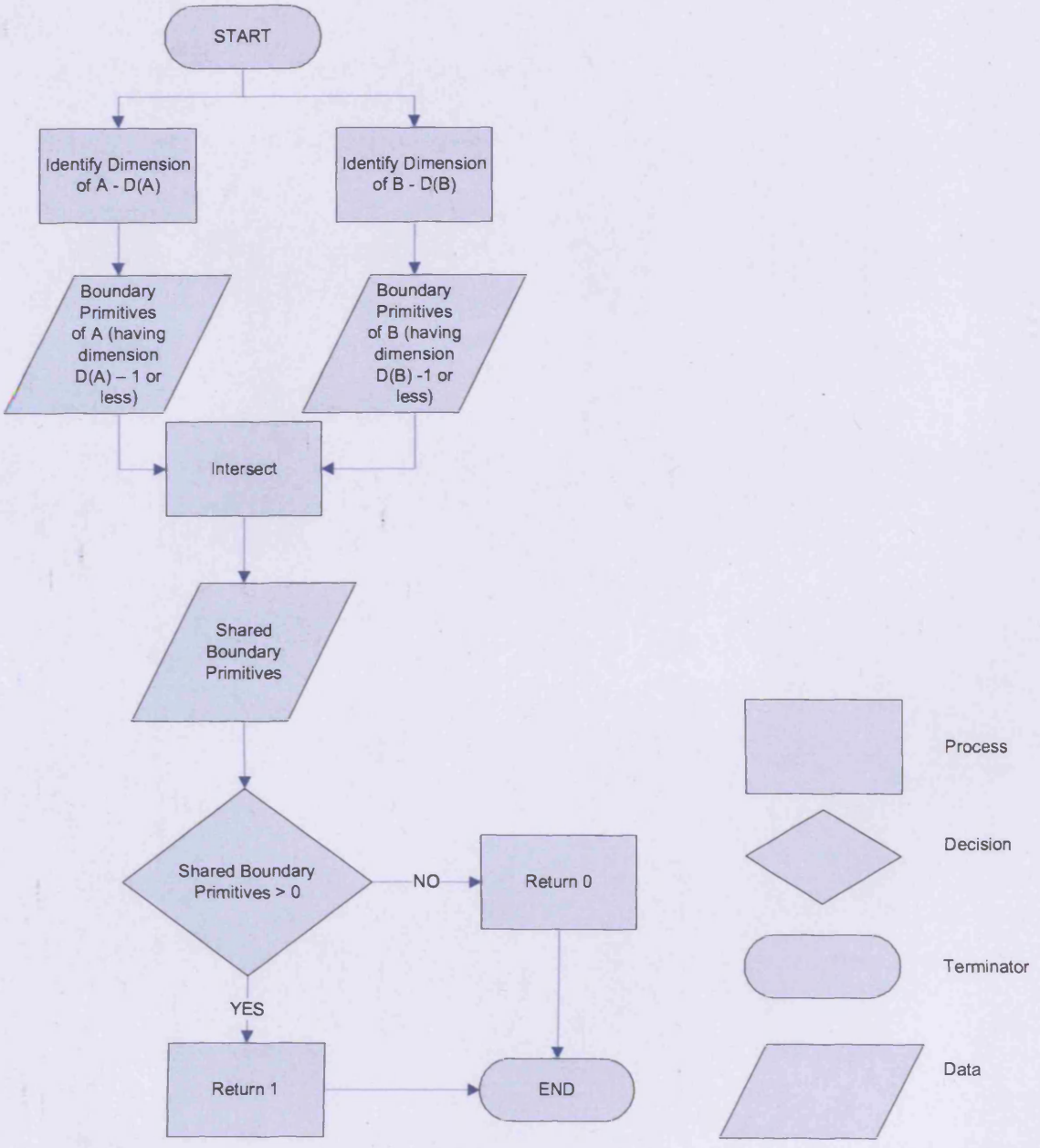


Figure 125 - Boundary Intersection between two Part Objects

Figure 125 shows the flow diagram for the determination of boundary intersection between two Part Objects. This function returns 1 if a boundary primitive of A is also a boundary primitive of B. This algorithm does not make an assumption that all objects will be related to boundary Nodes. Therefore, Nodes, Edges and Faces are all tested as part of this process as these can be boundary primitives. Where information identifying boundary primitives is not directly

encoded in the data structure (as it is for STS) the dimension of the boundary primitives is taken as any dimension less than the dimension of the Part Object itself. If A and B share boundary Nodes OR A and B share boundary Edges OR A and B share boundary Faces, then a value of 1 is returned by the function. The INTERSECT operation referenced is a set intersection.

Interior of Part Object A intersecting Interior of Part Object B

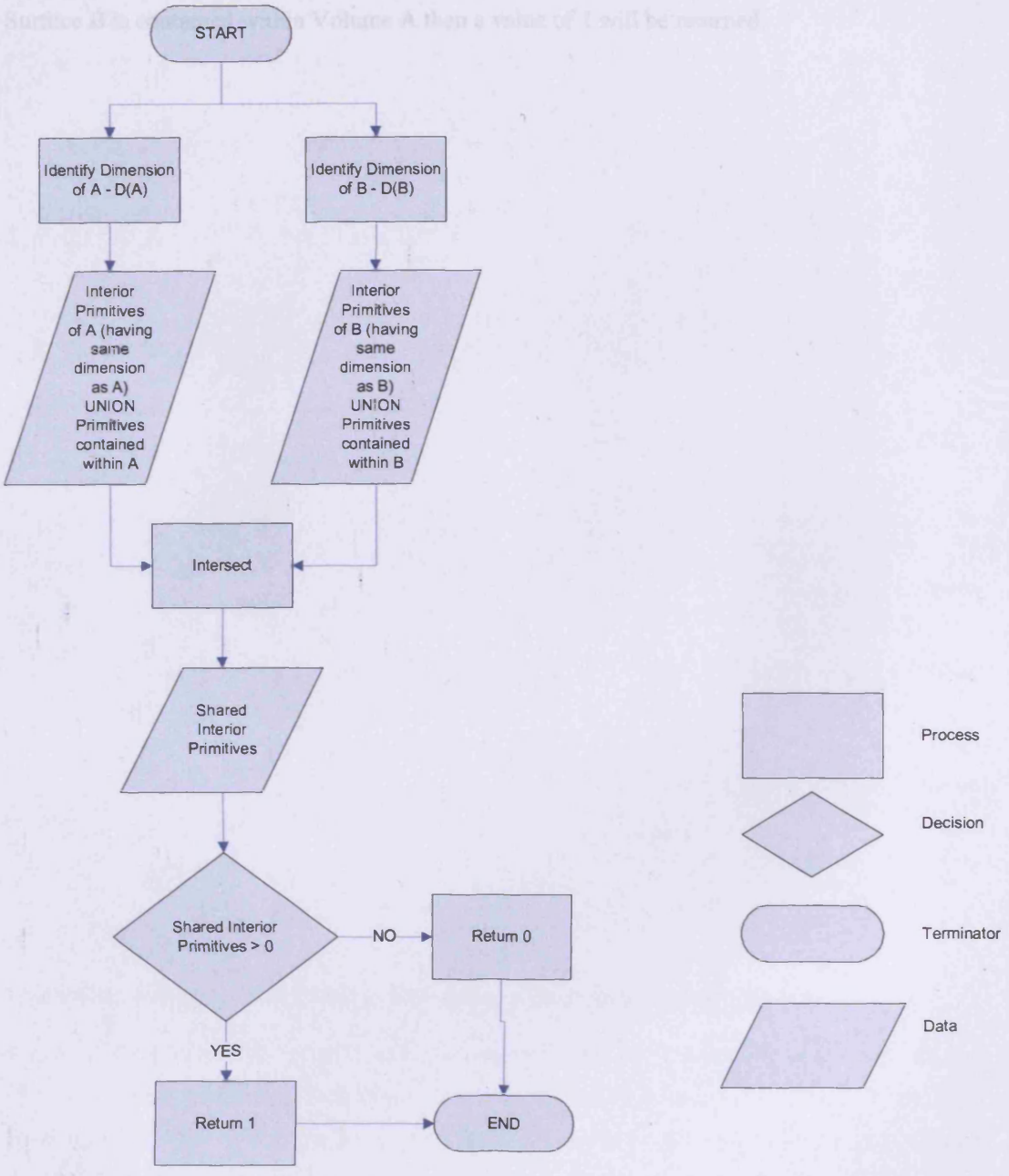


Figure 126 - Intersection of the Interiors of two Part Object

Figure 126 shows a flow diagram illustrating the process of identifying the interior intersection of two Part Objects. This function returns 1 if an interior primitive of A is also an interior primitive of B. The algorithm defines interior primitives as those having the same dimension as the Part Object, and initially determines the dimension of both Part Objects before identifying the interior primitives. Additionally, it searches for contained primitives of ANY dimension up

to that of the Part Object itself. If A and B share interior or contained Nodes OR A and B share interior or contained Edges OR A and B share interior or contained Faces or A and B share interior Volumes, then a value of 1 is returned by the algorithm. For example, if a Face of Surface B is contained within Volume A then a value of 1 will be returned.

Interior of Part Object A intersecting Exterior of Part Object B (and vice-versa)

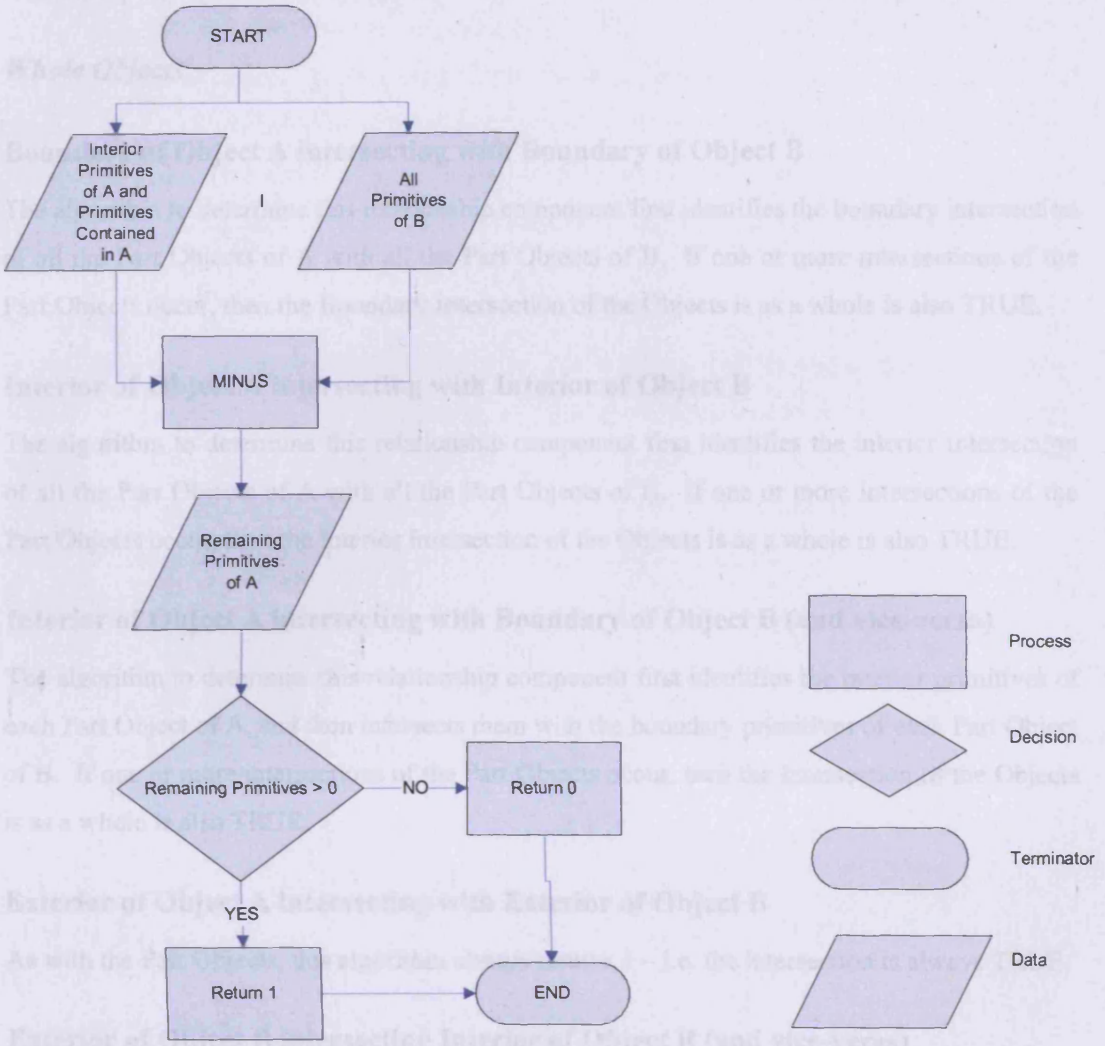


Figure 127 - Interior of Part Object A intersecting with Exterior of Part Object B

Figure 127 represents the process used to determine the intersection of the exterior of Part Object B with the interior of Part Object A or vice versa. This intersection returns 1 unless all the primitives forming the interior of Part Object A are also included in B (i.e. the MINUS operation returns an empty set). Interior primitives are defined as those having the same dimension as Part Object A. The algorithm also checks for contained primitives, having dimension up to that of Part Object A. This covers the situation where all of B is contained within A.

Exterior of Part Object A intersecting with Exterior of Part Object B

This is the exterior of A intersecting the exterior of B and is always TRUE for the binary relationship between two finite objects.

Whole Objects

Boundary of Object A intersecting with Boundary of Object B

The algorithm to determine this relationship component first identifies the boundary intersection of all the Part Objects of A with all the Part Objects of B. If one or more intersections of the Part Objects occur, then the Boundary Intersection of the Objects is as a whole is also TRUE.

Interior of Object A intersecting with Interior of Object B

The algorithm to determine this relationship component first identifies the interior intersection of all the Part Objects of A with all the Part Objects of B. If one or more intersections of the Part Objects occur, then the Interior Intersection of the Objects is as a whole is also TRUE.

Interior of Object A intersecting with Boundary of Object B (and vice-versa)

The algorithm to determine this relationship component first identifies the interior primitives of each Part Object of A, and then intersects them with the boundary primitives of each Part Object of B. If one or more intersections of the Part Objects occur, then the Intersection of the Objects is as a whole is also TRUE.

Exterior of Object A intersecting with Exterior of Object B

As with the Part Objects, this algorithm always returns 1 – i.e. the intersection is always TRUE.

Exterior of Object B intersecting Interior of Object B (and vice-versa)

If the interior of one Part of A is not shared with one Part of B, it cannot be inferred that it is not shared with ANY parts of B. Therefore the algorithm needs to validate ALL the interior components of the parts of A against B. The algorithm must therefore first determine the dimension of the Part of Object A being processed, and hence select any interior primitives for the Part Object. Primitives having the same dimension for each Part Object of B are then subtracted from the resulting list. If any subtractions return a non-empty set, then the interior of Object A intersects the Exterior of B.

Appendix 6 – Implementation

This Appendix gives an overview of the implementation of the 9-Intersection component relationship determination algorithms, describing each PL/SQL package forming part of the software tools developed. Each package is described in the form of a summary table and is first presented for STS, followed by the implementation for 3DFDS to facilitate comparison. Where appropriate, additional detail relating to the functions is also provided. The implementation of the algorithm to map user terminology to R-Values is given at the end of the appendix (Package TOPOLOGY).

In the PL/SQL implementation described in this appendix, all packages for the Extended 3DFDS structure are prefixed with the word GEN to differentiate them from STS packages having the same functionality. Additionally, table names in the 3DFDS structure are also prefixed with the word GEN. The term "object" refers to the SDO_GEOMETRY representation of each *FEATURE*. Italic block capitals refer to table and field names – for example *FEATURE*, *TOPO_PART_ID*.

The PL/SQL for a number of key aspects of the code has been included here. A full copy of all code can be found on the attached CD.

STS - Package NINE_INTERSECTIONS

A brief description of the functions contained within this package is given in Table 94 below.

Function Name	*Description*	*Parameters*	*Returns*
calc_geom_r_value	Initializes the call to determine all the component parts of the 9-Intersection relationship – breaks the object down into simple parts, identifies the 9-Intersection component value between the parts and determines the appropriate 9-Intersection component for the object as a whole	The two IDs of the objects The number (1 to 9) of the component of the intersection required	1 if an intersection exists, 0 if not
get_field_1	Determines the result of Boundary A intersect Boundary B for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_2	Determines the result of Interior A intersecting Interior B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_3	Determines the result of Boundary A intersecting Interior of B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_4	Determines the result of Interior of A intersecting Boundary of B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_5	Determines the result of Exterior A intersecting Exterior B, for PARTS of the objects	The two *TOPO_PART_IDs*	Always returns 1 – exteriors always intersect
get_field_6	Determines the result of Exterior A intersecting Boundary B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_7	Determines the result of Exterior A intersect Interior B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_8	Determines the result of Boundary A intersecting Exterior B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_9	Determines the result of Interior A intersect Exterior B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_geom_field_1	Calls CALC_GEOM_R_VALUE for Boundary A intersects Boundary B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_2	Calls CALC_GEOM_R_VALUE for Interior A intersecting Interior B, for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not

Sample code is shown below for get_field_4 and get_field_9:

```
FUNCTION get_field_4(parent_topo_id1 IN PLS_INTEGER, parent_topo_id2 IN PLS_INTEGER)
RETURN PLS_INTEGER
        IS
                sql_stmt VARCHAR2(2000);                dim_b PLS_INTEGER;
                num_shared_faces PLS_INTEGER;           num_shared_edges PLS_INTEGER;
                num_shared_nodes PLS_INTEGER;
        BEGIN   -- this is the reverse of get_field_3.  Find the dimension of the objct
so that we can determine the dimension of the  int and bnd primitives then try to
identify any shared faces (if dim = 3) edges  or nodes by running an inner join query on
the primitives.  If there are any shared primitives, then there is an intersection
                dim_b := shared_routines_9i.get_dimension(parent_topo_id2);
                IF dim_b = 3 THEN
                        sql_stmt := 'SELECT COUNT(*) FROM GEOM_FACE A INNER JOIN GEOM_FACE
B ON A.FACE_ID = B.FACE_ID AND B.PARENT_TOPO_ID = :1 AND B.IS_BOUNDARY = 0 AND
A.PARENT_TOPO_ID = :2 AND A.IS_BOUNDARY = 1';
                        EXECUTE IMMEDIATE sql_stmt INTO num_shared_faces USING
parent_topo_id1, parent_topo_id2;
                        sql_stmt := 'SELECT COUNT(*) FROM GEOM_EDGE A INNER JOIN GEOM_EDGE
B ON A.EDGE_ID = B.EDGE_ID AND B.PARENT_TOPO_ID = :1 AND B.IS_BOUNDARY = 0 AND
A.PARENT_TOPO_ID = :2 AND A.IS_BOUNDARY = 1';
                        EXECUTE IMMEDIATE sql_stmt INTO num_shared_edges USING
parent_topo_id1, parent_topo_id2;
                        sql_stmt := 'SELECT COUNT(*) FROM GEOM_NODE A INNER JOIN GEOM_NODE
B ON A.NODE_ID = B.NODE_ID AND B.PARENT_TOPO_ID = :1 AND B.IS_BOUNDARY = 0 AND
A.PARENT_TOPO_ID = :2 AND A.IS_BOUNDARY = 1';
                        EXECUTE IMMEDIATE sql_stmt INTO num_shared_nodes USING
parent_topo_id1, parent_topo_id2;
                        IF num_shared_faces > 0 OR num_shared_edges > 0 OR
num_shared_nodes > 0 THEN
                                RETURN 1;
                        ELSE
                                RETURN 0;
                        END IF;
                END IF;
                IF dim_b = 2 THEN
                        sql_stmt := 'SELECT COUNT(*) FROM GEOM_EDGE A INNER JOIN GEOM_EDGE
B ON A.EDGE_ID = B.EDGE_ID AND B.PARENT_TOPO_ID = :1 AND B.IS_BOUNDARY = 0 AND
A.PARENT_TOPO_ID = :2 AND A.IS_BOUNDARY = 1';
                        EXECUTE IMMEDIATE sql_stmt INTO num_shared_edges USING
parent_topo_id1, parent_topo_id2;
                        sql_stmt := 'SELECT COUNT(*) FROM GEOM_NODE A INNER JOIN GEOM_NODE
B ON A.NODE_ID = B.NODE_ID AND B.PARENT_TOPO_ID = :1 AND B.IS_BOUNDARY = 0 AND
A.PARENT_TOPO_ID = :2 AND A.IS_BOUNDARY = 1';
                        EXECUTE IMMEDIATE sql_stmt INTO num_shared_nodes USING
parent_topo_id1, parent_topo_id2;
                        IF num_shared_edges > 0 OR num_shared_nodes > 0 THEN
                                RETURN 1;
                        ELSE
                                RETURN 0;
                        END IF;
                END IF;
                IF dim_b = 1 THEN
                        sql_stmt := 'SELECT COUNT(*) FROM GEOM_NODE A INNER JOIN GEOM_NODE
B ON A.NODE_ID = B.NODE_ID AND B.PARENT_TOPO_ID = :1 AND B.IS_BOUNDARY = 0 AND
A.PARENT_TOPO_ID = :2 AND A.IS_BOUNDARY = 1';
                        EXECUTE IMMEDIATE sql_stmt INTO num_shared_nodes USING
parent_topo_id1, parent_topo_id2;
                        IF  num_shared_nodes > 0 THEN
                                RETURN 1;
                        ELSE
                                RETURN 0;
                        END IF;
                END IF;
END get_field_4;
FUNCTION get_field_9(parent_topo_id1 IN PLS_INTEGER, parent_topo_id2 IN PLS_INTEGER)
RETURN PLS_INTEGER
        IS
        BEGIN   -- THIS FIELD IS ALWASY TRUE IF YOU CANNOT recons A from the shared prims
                IF shared_routines_9I.reconstruct_a(parent_topo_id1,parent_topo_id2) =
'TRUE' THEN
                        RETURN 0;
                ELSE
                        RETURN 1;
                END IF;
END get_field_9;
```

Function Name	*Description*	*Parameters*	*Returns*
Get_geom_field_3	Calls CALC_GEOM_R_VALUE for Boundary A intersects Interior B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_4	Calls CALC_GEOM_R_VALUE for Interior A intersects Boundary B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_5	Calls CALC_GEOM_R_VALUE for Exterior A intersects Exterior B for the whole objects	The two *FEATURE_IDs*	Always returns 1
get_geom_field_6	Calls CALC_GEOM_R_VALUE for Exterior A intersecting Boundary B, for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_7	Calls CALC_GEOM_R_VALUE for Exterior A intersect Interior B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_8	Calls CALC_GEOM_R_VALUE for Boundary A intersecting Exterior B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_9	Calls CALC_GEOM_R_VALUE for Interior A intersect Exterior B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_r_value	Calculates the final R-Value for the 9-Intersection relationship for the whole objects, calling the GET_GEOM_FIELD_X functions in turn.	The two *FEATURE_IDs*	The R-Value
get_R_Value	Calculates the final R-Value for the 9-Intersection relationship for two given objects parts, calling the GET_FIELD_X functions in turn	The two *TOPO_PART_IDs*	The R-Value

Table 94 – Summary of Nine-Intersections Package

GET_FIELD_1 to GET_FIELD_9 determine the 9-Intersection relationship components for PART objects, returning 1 if an intersection occurs and 0 otherwise. GET_FIELD_1 to GET_FIELD_4 directly query the *TOPO_* tables to determine the interior or boundary primitives for each PART object, and then INTERSECT the results to determine shared primitives. GET_FIELD_5 (which represents the Exterior intersections) always returns 1. GET_FIELD_6 to GET_FIELD_9 define the Exterior of a PART object as anything that does not form part of the interior or boundary. Therefore to determine, for example, the relationship Exterior A intersecting Interior B, the *TOPO_* tables are queried for all non-boundary primitives of B, and for ALL primitives of A. The primitives of A are subtracted (MINUS) from the non-boundary primitives of B. If the resulting set is empty, then the function returns 0 (i.e. all the interior primitives of B also form part of A and hence are not exterior to A).

GET_GEOM_R_VALUE determines the R-Value for the relationship between the two given objects, by calling the GET_GEOM_FIELD_X routines to identify the individual components of the 9-Intersection relationship, and then multiplying the results as per the following formula:

```
THE_R_VALUE :=    GET_GEOM_FIELD_9 (GEOM_ID1, GEOM_ID2)          -- Int A intersect Ext B
            +     (GET_GEOM_FIELD_8 (GEOM_ID1, GEOM_ID2) * 2)    -- Bnd A intersect Ext B
            +     (GET_GEOM_FIELD_7 (GEOM_ID1, GEOM_ID2) * 4)    -- Ext A intersect Int B
            +     (GET_GEOM_FIELD_6 (GEOM_ID1, GEOM_ID2) * 8)    -- Ext A intersect Bnd B
            +     (GET_GEOM_FIELD_5 (GEOM_ID1, GEOM_ID2) * 16)   -- Ext A intersect Ext B
            +     (GET_GEOM_FIELD_4 (GEOM_ID1, GEOM_ID2) * 32)   -- Int A intersect Bnd B
            +     (GET_GEOM_FIELD_3 (GEOM_ID1, GEOM_ID2) * 64)   -- Bnd A intersect Int B
            +     (GET_GEOM_FIELD_2 (GEOM_ID1, GEOM_ID2) * 128)  -- Int A intersect Int B
            +     (GET_GEOM_FIELD_1 (GEOM_ID1, GEOM_ID2) * 256)  -- Bnd A intersect Bnd B
```

GET_R_VALUE - Calculates the final R-Value for the 9-Intersection relationship for two given Object PARTS, calling the GET_FIELD_X functions in turn. R-Value is calculated as per the function above, using shown in GET_GEOM_R_VALUE.

CALC_GEOM_R_VALUE Initializes the call to determine all the component parts of the 9-Intersection relationship – breaks the object down into simple parts, identifies the 9-Intersection component value between the parts and determines the appropriate 9-Intersection component for the object as a whole.

GET_GEOM_FIELD_X: These functions break the objects down into simple parts, identifying the 9-Intersection component value between the parts and determining the appropriate 9-Intersection component for the object as a whole, as per the rules described above.

3DFDS - Package GEN_NINE_INTERSECTIONS

A brief description of the functions contained within this package is given in Table 95 below. All functions implemented as for STS unless otherwise specified.

Function Name	*Description*	*Parameters*	*Returns*
calc_geom_r_value	Initializes the call to determine all the component parts of the 9-Intersection relationship – breaks the object down into simple parts, identifies the 9-Intersection component value between the parts and determines the appropriate 9-Intersection component for the object as a whole	The two IDs of the Features The number (1 to 9) of the component of the intersection required	1 if an intersection exists, 0 if not
get_field_1	Determines the result of Boundary A intersect Boundary B for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_2	Determines the result of Interior A intersecting Interior B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_3	Determines the result of Boundary A intersecting Interior of B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_4	Determines the result of Interior of A intersecting Boundary of B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_5	Determines the result of Exterior A intersecting Exterior B, for PARTS of the objects	The two *TOPO_PART_IDs*	Always returns 1 – exteriors always intersect
get_field_6	Determines the result of Exterior A intersecting Boundary B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_7	Determines the result of Exterior A intersect Interior B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_8	Determines the result of Boundary A intersecting Exterior B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_field_9	Determines the result of Interior A intersect Exterior B, for PARTS of the objects	The two *TOPO_PART_IDs*	1 if an intersection exists, 0 if not
get_geom_field_1	Calls CALC_GEOM_R_VALUE for Boundary A intersects Boundary B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_2	Calls CALC_GEOM_R_VALUE for Interior A intersecting Interior B, for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not

Function Name	*Description*	*Parameters*	*Returns*
get_geom_field_3	Calls CALC_GEOM_R_VALUE for Boundary A intersects Interior B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_4	Calls CALC_GEOM_R_VALUE for Interior A intersects Boundary B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_5	Calls CALC_GEOM_R_VALUE for Exterior A intersects Exterior B for the whole objects	The two *FEATURE_IDs*	Always returns 1
get_geom_field_6	Calls CALC_GEOM_R_VALUE for Exterior A intersecting Boundary B, for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_7	Calls CALC_GEOM_R_VALUE for Exterior A intersect Interior B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_8	Calls CALC_GEOM_R_VALUE for Boundary A intersecting Exterior B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_field_9	Calls CALC_GEOM_R_VALUE for Interior A intersect Exterior B for the whole objects	The two *FEATURE_IDs*	1 if an intersection exists, 0 if not
get_geom_r_value	Calculates the final R-Value for the 9-Intersection relationship for the whole objects, calling the GET_GEOM_FIELD_X functions in turn.	The two *FEATURE_IDs*	The R-Value
get_R_Value	Calculates the final R-Value for the 9-Intersection relationship for two given objects parts, calling the get_ field functions in turn	The two *TOPO_PART_IDs*	The R-Value

Table 95 – Functions included in GEN_NINE_INTERSECTIONS

GET_FIELD_X and GET_GEOM_FIELD_X functions are identical to those for STS. However, instead of directly querying the *TOPO_* tables to determine primitives associated with particular objects, the function calls SHARE_NODE.GET_ALL_NODES, SHARE_EDGE.GET_ALL_EDGES, SHARE_FACE.GET_ALL_FACES and SHARE_VOLUME.GET_ALL_VOLUMES. These functions return PL/SQL tables having the same structure as the *TOPO_* tables – i.e. with *PARENT_TOPO_ID*, *PRIMITIVE_ID* and *IS_BOUNDARY* flags.

STS - Package – SHARE_NODE

A brief description of the functions contained within this package is given in Table 96 below.

Function Name	*Description*	*Parameters*	*Returns*
detail_shared_nodes	Returns the IDs of any NODE primitives shared between the given PART objects	The *TOPO_PART_IDs*	PL/SQL table listing the NODE IDs

Table 96 - Functions included in SHARE_NODE

Queries the *TOPO_NODE* table using an INNER JOIN to return a list of Node IDs that are shared between the two Part Objects – and if the Nodes are boundary or interior Nodes for each Part Object. PL/SQL for this package is shown here:

```
create or replace type node_results_type as OBJECT (PARENT_TOPO_ID NUMBER(10), NODE_ID
NUMBER(10), IS_BOUNDARY NUMBER(1));
/
CREATE OR REPLACE TYPE node_results_table AS TABLE OF node_results_type;
/
-- use this type to return a list of IDs (either geometry or topology ids)
CREATE OR REPLACE TYPE number_results_type as OBJECT(ID NUMBER(10));
/
CREATE OR REPLACE TYPE number_table AS TABLE OF number_results_type;
/
CREATE OR REPLACE PACKAGE share_node AS
       FUNCTION detail_shared_nodes(parent_topo_id1 IN PLS_INTEGER, parent_topo_id2 IN
PLS_INTEGER) RETURN node_results_table PIPELINED;
END share_node;
/
CREATE OR REPLACE PACKAGE BODY share_node AS
-- use an inner join query on the GEOM_NODE table to find any shared nodes between the
two part objects and return these as a pipelined table
       FUNCTION detail_shared_nodes(parent_topo_id1 IN PLS_INTEGER, parent_topo_id2 IN
PLS_INTEGER) RETURN node_results_table PIPELINED
       IS
              sql_stmt VARCHAR2(200);
              TYPE NodeCurTyp IS REF CURSOR;
              node_cv NodeCurTyp;
              node_rec GEOM_NODE%ROWTYPE;
       BEGIN
              sql_stmt := 'SELECT A.PARENT_TOPO_ID, A.NODE_ID, A.IS_BOUNDARY FROM
GEOM_NODE A INNER JOIN GEOM_NODE B ON A.NODE_ID = B.NODE_ID WHERE B.PARENT_TOPO_ID = :1
AND A.PARENT_TOPO_ID = :2';
              OPEN node_cv FOR sql_stmt USING parent_topo_id1, parent_topo_id2;
              LOOP
                 FETCH node_cv INTO node_rec;
                     EXIT WHEN node_cv%NOTFOUND;
                     pipe row (node_results_type(node_rec.PARENT_TOPO_ID,
node_rec.NODE_ID,node_rec.IS_BOUNDARY));
              END LOOP;
              CLOSE node_cv;
       END detail_shared_nodes;
END share_node;
/
```

3DFDS - Package GEN_SHARE_NODE

A brief description of the functions contained within this package is given in Table 97 below.

Function Name	*Description*	*Parameters*	*Returns*
detail_shared_nodes	Lists the NODE PRIMITIVES of part object A and intersects this list with the NODE primitives of part object B	The *TOPO_PART_IDs*	PL/SQL table of the shared NODE IDs
get_all_nodes	Calls get_all_nodes_prep and get_all_nodes_ex_prep	The *TOPO_PART_ID*	PL/SQL table of the NODE IDs, and whether they are boundary or interior to the Part object
get_all_nodes_prep	Identifies any NODE primitives associated with the part object through the main 3DFDS data structure (i.e. from the GEN_NODE table)	The *TOPO_PART_ID*	PL/SQL table of the NODE IDs, and whether they are boundary or interior to the PART object
get_all_node_ex_prep	Identifies any NODE primitives associated with the part object that can be determined through the EXCEPTION tables in the structure	The *TOPO_PART_ID*	PL/SQL table of the NODE IDs, and whether they are boundary or interior to the PART object

Table 97 – Functions included in GEN_SHARE_NODES

As with STS, the DETAIL_SHARED_NODES function finds all the Nodes that are associated with each Part Object, using the SHARE_NODE.GET_ALL_NODES function. GET_ALL_NODES returns a PL/SQL table identical to that resulting from a query on the *TOPO_NODE* table for STS (i.e. having *PARENT_TOPO_ID*, *NODE_ID* and *IS_BOUNDARY* fields). Node primitives associated with objects are stored in three locations – the NODE table, for the Nodes forming part of the object itself, and the *NODE_ON_FACE* table and the *NODE_IN_VOLUME* table (depending on the dimension of the object) for any Nodes contained on the surface or within the object (i.e. containment exceptions). This routine calls the GET_ALL_NODES_PREP function to identify any non-exception Nodes, and the GET_ALL_NODES_EX_PREP function to identify exception Nodes.

GET_ALL_NODES_PREP: The number of joins required to identify Node primitives associated with an object depends on the dimension of the object in question. For example, to find the Nodes related to a Body object joins between the *VOLUME*, *FACE*, *EDGE* and *NODE* primitives must be followed. However, for a line object, only joins between the EDGE and NODE primitives are required. Therefore Step 1 of this algorithm is the determination of the dimension of the object (using SHARED_ROUTINES_9I.GET_DIMENSION)

If the dimension is 3 then the algorithm follows joins from the *FEATURE* table to the *TOPO_PART_TABLE* to the *TOPO_VOLUME* table to the *FACE_VOLUME* table to the *EDGE_FACE* table to the *NODE_EDGE* table to retrieve the required Nodes. The algorithm ignores any Nodes that result from Faces contained within the Volume, as these are listed as containment exceptions in the *NODE_IN_VOL* table. This is achieved by counting the number of times a Face is associated with the object. If the Face appears twice, then it is contained within the Volume and should be ignored. All Nodes returned by this function are boundary Nodes for the 3D object. If the dimension is 2 then all returned Nodes are again boundary Nodes. A similar join process is followed to retrieve the required Nodes, starting from the *TOPO_FACE* table. If the dimension is 1 then Nodes are again boundary Nodes unless they appear twice in the list returned by the join query, in which case they represent the interior of the line. If the dimension is 0, all required Nodes can be queried directly from the *TOPO_NODE* table.

GET_ALL_NODES_EX_PREP: This function returns any EXCEPTION Nodes for an object. Node exceptions can be of two kinds - Nodes in Volumes, and Nodes on Faces. The existence of such exceptions again depends on the dimension of the object in question, which must be queried at the start of the function using the GEN_SHARED_ROUTINES_9I.GET_DIMENSION function. If the dimension is 3 then the object can have both Node in Volume exceptions (these should be flagged as non-boundary) and have Node-on-Face exceptions. As the Face forms part of the boundary of the 3D object, such Nodes should be flagged as boundary Nodes. If dimension of the object is 2 then there are no Node-in-Volume exceptions.

Additionally, this function checks for any Node exceptions that arise where all the Edges linked to a Nodes are exceptions in a Face. These are identified by determining all the non-boundary Edges of the surface (using SHARE_EDGE.DETAIL_SHARED_EDGES) and then querying the NODE_EDGE table to identify all Nodes associated with these Edges. The result is subtracted from any Nodes that are also associated with boundary Edges of the Face.

PL/SQL for the GEN_SHARE_NODE package is given below, and can be contrasted with SHARE_NODE above.

```
CREATE OR REPLACE TYPE node_results_type AS OBJECT (
   parent_topo_id   NUMBER (10),
   node_id          NUMBER (10),
   is_boundary      NUMBER (1)
);
/
CREATE OR REPLACE TYPE node_results_table AS TABLE OF node_results_type;
/
-- use this type to return a list of IDs (either geometry or topology ids)
CREATE OR REPLACE TYPE number_results_type AS OBJECT (
   ID   NUMBER (10)
);
/
CREATE OR REPLACE TYPE number_table AS TABLE OF number_results_type;
/
CREATE OR REPLACE PACKAGE gen_share_node
AS
   FUNCTION detail_shared_nodes (
      parent_topo_id1   IN   PLS_INTEGER,
      parent_topo_id2   IN   PLS_INTEGER
   )
      RETURN node_results_table PIPELINED;
  FUNCTION get_all_nodes (parent_topo_id1 IN PLS_INTEGER)
      RETURN node_results_table PIPELINED;

   FUNCTION get_all_nodes_prep (parent_topo_id1 IN PLS_INTEGER)
      RETURN node_results_table PIPELINED;

   FUNCTION get_all_nodes_ex_prep (parent_topo_id1 IN PLS_INTEGER)
      RETURN node_results_table PIPELINED;
END gen_share_node;
/
CREATE OR REPLACE PACKAGE BODY gen_share_node
AS
   FUNCTION detail_shared_nodes (
      parent_topo_id1   IN   PLS_INTEGER,
      parent_topo_id2   IN   PLS_INTEGER
   )
      RETURN node_results_table PIPELINED
   IS
      sql_stmt     VARCHAR2 (200);
      sql_stmt_1   VARCHAR2 (200);
      sql_stmt_2   VARCHAR2 (200);

      TYPE nodecurtyp IS REF CURSOR;

      node_cv      nodecurtyp;
      node_id      PLS_INTEGER;
-- to find the share nodes, call the get_all_nodes subroutine for each part feature
-- then pipe them out
   BEGIN
      sql_stmt_1 :=
         'SELECT A.NODE_ID FROM TABLE(GEN_SHARE_NODE.GET_ALL_NODES(:1)) A INNER JOIN ';
      sql_stmt_2 :=
         '(SELECT NODE_ID FROM TABLE(GEN_SHARE_NODE.GET_ALL_NODES(:2))) B ON A.NODE_ID =
B.NODE_ID ';
      sql_stmt := sql_stmt_1 || sql_stmt_2;
      OPEN node_cv FOR sql_stmt USING parent_topo_id1, parent_topo_id2;
      LOOP
         FETCH node_cv
          INTO node_id;
         EXIT WHEN node_cv%NOTFOUND;
         -- nb we don't need the parent topo id or the boundary as this may differ
between the geometries! so use 0 and -1
         PIPE ROW (node_results_type (0, node_id, -1));
      END LOOP;

      CLOSE node_cv;
   END detail_shared_nodes;

-- the queries to get all the nodes associated with a particular part object
-- need to query the nodes table (following the required number of joins to identify
-- the nodes associated with the feature, depending on the dimension of the feature
-- additionally, the exception tables need to be queried just in case a node of one part
feature is an exception of the other (i.e. a topological relationship does exist between
them)
```

```
   FUNCTION get_all_nodes (parent_topo_id1 IN PLS_INTEGER)
      RETURN node_results_table PIPELINED
   IS
      sql_stmt       VARCHAR2 (1000);
      TYPE nodecurtyp IS REF CURSOR;
      node_cv        nodecurtyp;
      node_id        PLS_INTEGER;
      is_boundary    PLS_INTEGER;
   BEGIN
      sql_stmt :=
         'SELECT NODE_ID, is_boundary FROM
TABLE(GEN_SHARE_NODE.GET_ALL_NODES_PREP(:1))';
      OPEN node_cv FOR sql_stmt USING parent_topo_id1;
      LOOP
         FETCH node_cv
          INTO node_id, is_boundary;

         EXIT WHEN node_cv%NOTFOUND;
         PIPE ROW (node_results_type (parent_topo_id1, node_id, is_boundary));
      END LOOP;

      CLOSE node_cv;
      sql_stmt :=
         'SELECT NODE_ID, IS_BOUNDARY FROM
TABLE(GEN_SHARE_NODE.GET_ALL_NODES_ex_PREP(:1)) ';
      OPEN node_cv FOR sql_stmt USING parent_topo_id1;
      LOOP
         FETCH node_cv
          INTO node_id, is_boundary;
         EXIT WHEN node_cv%NOTFOUND;
         PIPE ROW (node_results_type (parent_topo_id1, node_id, is_boundary));
      END LOOP;
      CLOSE node_cv;
   END get_all_nodes;
   FUNCTION get_all_nodes_ex_prep (parent_topo_id1 IN PLS_INTEGER)
      RETURN node_results_table PIPELINED
   IS
      sql_stmt           VARCHAR2 (1000);
      TYPE nodecurtyp IS REF CURSOR;
      node_cv            nodecurtyp;
      the_dimension_1    PLS_INTEGER;
      sql_stmt_1         VARCHAR2 (1000);
      sql_stmt_2         VARCHAR2 (1000);
      sql_stmt_3         VARCHAR2 (200);
      sql_stmt_4         VARCHAR2 (200);
      node_id            PLS_INTEGER;
      is_boundary        PLS_INTEGER;
      parent_topo_id     PLS_INTEGER;
   BEGIN
      -- GET ALL THE exception NODES, depending on the dimension of the object
      the_dimension_1 :=
                        gen_shared_routines_9i.get_dimension (parent_topo_id1);
      IF the_dimension_1 = 3
      THEN
-- ONLY GET THE VOLUME EXCEPTIONS WITH IS_BOUNDARY = 0, THE FACE_EXCEPTIONS SHOULD BE
MARKED IS_BOUNDARY = 1
         sql_stmt_2 :=
            'SELECT :1 AS PARENT_TOPO_ID,NODE_ID,0 AS IS_BOUNDARY FROM GEN_NODE_VOL_EX A
INNER JOIN GEN_GEOM_VOLUME B ON A.VOLUME_ID = B.VOLUME_ID AND B.PARENT_TOPO_ID = :2';
         sql_stmt_3 :=
            ' UNION SELECT :3 AS PARENT_TOPO_ID, NODE_ID,1 AS IS_BOUNDARY FROM
GEN_NODE_FACE_EX A INNER JOIN (SELECT FACE_ID FROM GEN_FACE_VOLUME A INNER JOIN
GEN_GEOM_VOLUME B';
         sql_stmt_4 :=
            ' ON A.VOLUME_ID = B.VOLUME_ID AND B.PARENT_TOPO_ID = :4) C ON A.FACE_ID =
C.FACE_ID';
         sql_stmt_2 := sql_stmt_2 || sql_stmt_3 || sql_stmt_4;
         sql_stmt := sql_stmt_2;

         OPEN node_cv FOR sql_stmt
         USING parent_topo_id1,
               parent_topo_id1,
               parent_topo_id1,
               parent_topo_id1;
         LOOP
            FETCH node_cv
```

```
            INTO parent_topo_id, node_id, is_boundary;
            EXIT WHEN node_cv%NOTFOUND;
            PIPE ROW (node_results_type (parent_topo_id1,
                                         node_id,
                                         is_boundary
                                        ));
         END LOOP;
         CLOSE node_cv;
      END IF;
      IF the_dimension_1 = 2
      THEN
         -- THERE WILL NOT BE ANY NODE/VOLUME EXCEPTIONS
         sql_stmt_2 :=
            'SELECT :1 AS PARENT_TOPO_ID,NODE_ID,0 AS IS_BOUNDARY FROM GEN_NODE_FACE_EX
A INNER JOIN GEN_GEOM_FACE B ON A.FACE_ID = B.FACE_ID AND B.PARENT_TOPO_ID = :2';
         sql_stmt := sql_stmt_2;
         OPEN node_cv FOR sql_stmt USING parent_topo_id1, parent_topo_id1;
         LOOP
            FETCH node_cv
             INTO parent_topo_id, node_id, is_boundary;
            EXIT WHEN node_cv%NOTFOUND;
            PIPE ROW (node_results_type (parent_topo_id1,
                                         node_id,
                                         is_boundary
                                        ));
         END LOOP;
         CLOSE node_cv;
      END IF;
      IF the_dimension_1 = 2
      THEN
         -- WE ALSO NEED TO CHECK FOR NODE EXCEPTIONS THAT ARISE BECAUSE ALL THE
CONNECTING EDGES ARE ALSO EXCEPTIONS
         -- FOR THE NON_BOUNDARY NODES
         -- use 2 as the code for non-boundary
         sql_stmt_1 :=
            ' SELECT :1 as parent_topo_id, NODE_ID, 0 as is_boundary FROM ( ';
         sql_stmt_1 :=
               sql_stmt_1
            || 'select DISTINCT A.NODE_ID, B.IS_BOUNDARY from gen_node_edge A inner join
table(gen_share_edge.get_all_edges(:2)) B ';
         sql_stmt_1 :=
               sql_stmt_1
            || ' ON A.EDGE_ID = B.EDGE_ID WHERE IS_BOUNDARY = 0  and node_id not in
(select DISTINCT A.NODE_ID from gen_node_edge A ';
         sql_stmt_1 :=
               sql_stmt_1
            || 'inner join table(gen_share_edge.get_all_edges(:3)) B ON A.EDGE_ID =
B.EDGE_ID WHERE IS_BOUNDARY = 1))';
         OPEN node_cv FOR sql_stmt_1
         USING parent_topo_id1, parent_topo_id1, parent_topo_id1;
         LOOP
            FETCH node_cv
             INTO parent_topo_id, node_id, is_boundary;

            EXIT WHEN node_cv%NOTFOUND;
            PIPE ROW (node_results_type (parent_topo_id1,
                                         node_id,
                                         is_boundary
                                        ));
         END LOOP;
         CLOSE node_cv;
      END IF;
   END get_all_nodes_ex_prep;
   FUNCTION get_all_nodes_prep (parent_topo_id1 IN PLS_INTEGER)
      RETURN node_results_table PIPELINED
   IS
      sql_stmt          VARCHAR2 (1000);
      TYPE nodecurtyp IS REF CURSOR;
      node_cv           nodecurtyp;
      the_dimension_1   PLS_INTEGER;
      the_dimension_2   PLS_INTEGER;
      sql_stmt_1        VARCHAR2 (1000);
      sql_stmt_2        VARCHAR2 (1000);
      sql_stmt_3        VARCHAR2 (200);
      sql_stmt_4        VARCHAR2 (200);
      node_id           PLS_INTEGER;
```

```
      is_boundary       PLS_INTEGER;
      parent_topo_id    PLS_INTEGER;
      num_node          PLS_INTEGER;
   BEGIN
      -- GET ALL THE NODES, not from exception tables
      -- if dimension = 3 or 2 then nodes are all is_boundary = 1
      -- if dimension = 3 and nodes appear TWICE don't output them as they are
exceptions on a face inside the volume (nodes shoudl appear 3 times for a volume)
      -- if dimension = 1 then if nodes appear twice they are is_boundary = 0
      -- if a node is referenced ONCE on a face or in a volume then don't send out as
this is an exception node and will be handled in exception routine
      the_dimension_1 :=
                        gen_shared_routines_9i.get_dimension (parent_topo_id1);
      DBMS_OUTPUT.put_line (the_dimension_1);
      DBMS_OUTPUT.put_line (the_dimension_2);
      IF the_dimension_1 = 3
      THEN
         sql_stmt_1 :=
            'SELECT DISTINCT F.NUM_FACES as num_node, :1  AS PARENT_TOPO_ID, NODE_ID, 1
AS IS_BOUNDARY  FROM GEN_NODE_EDGE G INNER JOIN ( ';
         sql_stmt_2 :=
            ' SELECT DISTINCT D.EDGE_ID, E.NUM_FACES FROM GEN_EDGE_FACE D INNER JOIN
(SELECT COUNT(*) AS NUM_FACES, FACE_ID FROM GEN_FACE_VOLUME A INNER JOIN GEN_GEOM_VOLUME
B ';
         sql_stmt_3 :=
            ' ON A.VOLUME_ID = B.VOLUME_ID AND B.PARENT_TOPO_ID = :1 GROUP BY FACE_ID) E
';
         sql_stmt_4 :=
                        ' ON E.FACE_ID = D.FACE_ID) F ON F.EDGE_ID = G.EDGE_ID';
         sql_stmt_1 := sql_stmt_1 || sql_stmt_2 || sql_stmt_3 || sql_stmt_4;
      END IF;
      IF the_dimension_1 = 2
      THEN
         -- NEED TO DETECT THESE BY CHECKING TO SEE IF ALL THE EDGES TOUCHING THOSE
NODES ARE ALSO NON_BOUNDARY
         -- FOR THE BOUNDARY NODES
         -- non boundary nodes are handled in the _ex section
         -- use 0 as the code for boundary nodes
         sql_stmt_1 :=
            'SELECT 0 as num_node,:1 as parent_topo_id, node_id, 1 as is_boundary FROM
(select DISTINCT A.NODE_ID, B.IS_BOUNDARY from gen_node_edge A inner join
table(gen_share_edge.get_all_edges(:2)) B ';
         sql_stmt_1 :=
              sql_stmt_1 || 'ON A.EDGE_ID = B.EDGE_ID) WHERE IS_BOUNDARY = 1';
      END IF;
      IF the_dimension_1 = 1
      THEN
         sql_stmt_1 :=
            ' SELECT COUNT(*) as num_node, :1 AS PARENT_TOPO_ID , NODE_ID,1 AS
IS_BOUNDARY FROM GEN_NODE_EDGE E INNER JOIN (SELECT EDGE_ID FROM GEN_GEOM_EDGE D ';
         sql_stmt_1 :=
               sql_stmt_1
            || ' WHERE  D.PARENT_TOPO_ID = :2) C ON C.EDGE_ID = E.EDGE_ID group by
node_id';
      END IF;
      IF the_dimension_1 = 0
      THEN
         sql_stmt_1 :=
            ' SELECT count(*) as num_node, :1 AS PARENT_TOPO_ID, NODE_ID,1 AS
IS_BOUNDARY FROM GEN_GEOM_NODE WHERE PARENT_TOPO_ID = :2 group by node_id';
      END IF;
      sql_stmt := sql_stmt_1;
      OPEN node_cv FOR sql_stmt USING parent_topo_id1, parent_topo_id1;
      LOOP
         -- nb for volume objects num_node is actually the number of faces!
         FETCH node_cv
          INTO num_node, parent_topo_id, node_id, is_boundary;
         is_boundary := 1;
         EXIT WHEN node_cv%NOTFOUND;
         IF     NOT (the_dimension_1 = 2 AND num_node = 1)
            AND NOT (the_dimension_1 = 3 AND num_node > 1)
         THEN
            IF (the_dimension_1 = 2 AND num_node = 2)
            THEN
               is_boundary := 0;
            -- due to all the edges also being exceptiosn on the surface
```

```
            END IF;
            IF (the_dimension_1 = 1 AND num_node = 2)
            THEN
               is_boundary := 0;
            END IF;
            PIPE ROW (node_results_type (parent_topo_id1, node_id,
                                         is_boundary));
         END IF;
      END LOOP;
      CLOSE node_cv;
   END get_all_nodes_prep;
END gen_share_node;
/
```

STS - Package – SHARE_EDGE

A brief description of the functions contained within this package is given in Table 98 below.

Function Name	*Description*	*Parameters*	*Returns*
detail_shared_edges	Returns the IDs of any EDGE primitives shared between the given PART objects	The *TOPO_PART_IDs*	PL/SQL table listing the EDGE IDs

Table 98 - Functions included in SHARE_EDGE

DETAIL_SHARED_EDGES Runs an inner join query on the *TOPO_EDGE* table (using an INNER JOIN) to find any Edges that are associated with both Part Object A and Part Object B.

3DFDS - Package GEN_SHARE_EDGE

A brief description of the functions contained within this package is given in Table 99 below. All functions implemented as for STS unless otherwise specified.

Function Name	*Description*	*Parameters*	*Returns*
detail_shared_edges	Lists the EDGE PRIMITIVES of part object A and intersects this list with the EDGE primitives of part object B	The *TOPO_PART_IDs*	PL/SQL table of the shared EDGE IDs
get_all_edges	Calls get_all_edges_prep and get_all_edges_ex_prep	The *TOPO_PART_ID*	PL/SQL table of the EDGE IDs, and whether they are boundary or interior to the PART OBJECT
get_all_edges_prep	Identifies any EDGE primitives associated with the part object through the main 3DFDS data structure (i.e. from the GEN_EDGE table)	The *TOPO_PART_ID*	PL/SQL table of the EDGE IDs, and whether they are boundary or interior to the part object
get_all_edge_ex_prep	Identifies any EDGE primitives associated with the part object that can be determined through the EXCEPTION tables in the structure	The *TOPO_PART_ID*	PL/SQL table of the EDGE IDs, and whether they are boundary or interior to the PART OBJECT

Table 99 – Functions included in GEN_SHARE_EDGE

DETAIL_SHARED_EDGES Finds all the Edges that are associated with each Part Object, using the SHARE_EDGE.GET_ALL_EDGES function. Identify any common Edges using an INTERSECT query.

GET_ALL_EDGES returns a PL/SQL table identical to that resulting from a query on the TOPO_NODE table for STS (i.e. having *PARENT_TOPO_ID*, *NODE_ID* and *IS_BOUNDARY* fields). It calls the GET_ALL_EDGES_PREP and GET_ALL_EDGES_EX_PREP functions.

GET_ALL_EDGES_PREP returns all Edges directly associated with a Part Object (i.e. not through containment exceptions). As the path to determine Edges associated with a Part Object depends on the dimension of the Part Object, the function initially calls the GEN_SHARED_ROUTINES_9I.GET_DIMENSION function. The boundary flag for Edges is determined by the dimension of the Part Object itself – Edges are boundary primitives for Surface and Body objects.

- If the dimension is 3 then the Edges are identified by following through from the TOPO_VOLUME table to FACE_VOLUME and hence to EDGE_FACE. Note that in this case, if an Edge is returned more than once by the query, this is due to the fact that the Face is contained in the Volume. The Edge is thus a containment exception and is ignored by the pipelined object, as it will be handled by GET_ALL_EDGES_EX_PREP.
- If the dimension is 2, a similar query to that above is run, following joins from the TOPO_FACE table. In this case if an Edge is returned twice, then it forms part of an inner Face of the surface object (an application of the Poincare Simplicial Homology described by Penninga 2007), and is thus not a boundary Edge.
- If the dimension is 1, associated Edges can be determined by directly querying the TOPO_EDGE table.
- If the dimension = 0, there will not be any Edges associated with the object.

GET_ALL_EDGES_EX_PREP returns any exception Edges in 3D and 2D objects. Note that this routine does not handle the case where an Edge is contained INSIDE a Face which is in turn INSIDE a Volume. Again, the presence of certain exception types depends on the dimension of the Part Object. Thus the routine first calls GEN_SHARED_ROUTINES_9I.GET_DIMENSION. If dimension is 3, the EDGE_IN_VOLUME exception table is queried, as well as the *EDGE_IN_FACE* table. If dimension is 2, then the *EDGE_IN_FACE* table is queried.

STS - Package – SHARE_FACE

A brief description of the functions contained within this package is given in Table 100 below.

Function Name	*Description*	*Parameters*	*Returns*
detail_shared_faces	Returns the IDs of any FACE primitives shared between the given PART objects	The *TOPO_PART_IDs*	PL/SQL table listing the FACE IDs

Table 100 - Functions included in SHARE_FACE

DETAIL_SHARED_FACES runs an INNER JOIN query on the *TOPO_FACE* table to find any Faces that are associated with both Part Object A and Part Object B.

3DFDS - Package GEN_SHARE_FACE

A brief description of the functions contained within this package is given in Table 101 below. All functions implemented as for STS unless otherwise specified.

Function Name	*Description*	*Parameters*	*Returns*
detail_shared_faces	Lists the FACE PRIMITIVES of part object A and intersects this list with the FACE primitives of part object B	The *TOPO_PART_IDs*	PL/SQL table of the shared FACE IDs
get_all_faces	Returns the FACE primitives associated with a part object	The *TOPO_PART_ID*	PL/SQL table of the FACE IDs

Table 101 – Functions included in GEN_SHARE_FACE

DETAIL_SHARED_FACES finds all the Faces that are associated with each object part, using the SHARE_FACE.GET_ALL_FACES query, and then intersects the results of each query.

GET_ALL_FACES: In the case of Face primitives, there are no possible exceptions to consider. Again, the boundary flag for each primitive will depend on the dimension of the object Part, which is queried using the GEN_SHARED_ROUTINES_9I.GET_DIMENSION function.

- If the, dimension is 3, then any associated Faces are boundary Faces, and are identified by querying the *TOPO_VOLUME* table and hence the *FACE_VOLUME* table. Note that any Faces returned more than once are, however, not boundary Faces – these are contained within the Volume in question.
- If the dimension is 2, then all associated Faces are non-boundary, and can be identified directly from the *TOPO_FACE* table.
- If the dimension is 1 or the dimension is 0 then no Faces are returned.

STS - Package – SHARE_VOLUME

A brief description of the functions contained within this package is given in Table 102 below.

Function Name	*Description*	*Parameters*	*Returns*
detail_shared_volumes	Returns the IDs of any VOLUME primitives shared between the given PART objects	The *TOPO_PART_IDs*	PL/SQL table listing the Volume IDs

Table 102 - Functions included in SHARE_VOLUME

DETAIL_SHARED_VOLUMES runs an INNER JOIN query on the *TOPO_VOLUME* table to find any Faces that are associated with both Part Object A and Part Object B.

3DFDS - Package GEN_SHARE_VOLUME

A brief description of the functions contained within this package is given in Table 103 below. Unless otherwise specified, all functions are implemented as for STS.

Function Name	*Description*	*Parameters*	*Returns*
detail_shared_non_ex_volumes	Lists the VOLUME PRIMITIVES of part object A and intersects this list with the VOLUME primitives of part object B	The *TOPO_PART_IDs*	PL/SQL table of the shared VOLUME IDs
detail_shared_volumes	Calls detail_shared_non_ex_volumes	The *TOPO_PART_IDs*	PL/SQL table of the shared VOLUME IDs
get_all_volumes	Returns the VOLUME primitives associated with a part object	The *TOPO_PART_ID*	PL/SQL table of the VOLUME IDs

Table 103 – Functions included in GEN_FIND_INTERSECTING_OBJECTS

DETAIL_SHARED_VOLUMES Finds all the Volumes that are associated with each object using the SHARE_VOLUME.GET_ALL_VOLUMES query, then find the intersection of the results of each query.

GET_ALL_VOLUMES: As with GET_ALL_FACES, there are no exceptions to consider here. If the dimension of the object Part is 3, then Volumes can be identified directly from the *TOPO_VOLUME* table. No Volume primitives are associated with objects of dimension < 3.

STS - Packages – SHARED_ROUTINES_9I

A brief description of the functions contained within this package is given in Table 104.

Function Name	*Description*	*Parameters*	*Returns*
get_dimension	Determines the maximum dimension of the primitives for a part object	The *TOPO_PART_ID*	0, 1, 2 or 3
reconstruct_a	Determines PART object A can be reconstructed from the primitives it shares with PART object B	The *TOPO_PART_IDs*	TRUE or FALSE
reconstruct_b	Determines if PART object B can be reconstructed from the primitives it shares with PART object A	The *TOPO_PART_IDs*	TRUE or FALSE

Table 104 – Functions contained within SHARED_ROUTINES_9I

RECONSTRUCT_A determines whether it is possible to reconstruct Part Object A from the primitives shared with Part Object B. The primitive IDs making up A are subtracted from the primitive of the same dimension making up B (calling the DETAIL_SHARED_NODES, DETAIL_SHARED_EDGES, DETAIL_SHARED_FACE and DETAIL_SHARED_VOLUME

functions to identify the required primitives). If ALL the remaining totals are 0 then it is possible to reconstruct A from the shared primitives (as all the primitives of A are shared with B).

GET_DIMENSION determines the dimension of the Part Object by querying the *TOPO_NODE*, *TOPO_EDGE*, *TOPO_FACE* and *TOPO_VOLUME* tables to identify the highest dimension primitive. It returns the highest dimension.

3DFDS - Package GEN_SHARED_ROUTINES_9I

A brief description of the functions contained within this package is given in Table 105 below. All functions are as described for STS.

Function Name	*Description*	*Parameters*	*Returns*
get_dimension	Determines the maximum dimension of the primitives for a part object	The *TOPO_PART_ID*	0, 1, 2 or 3
reconstruct_a	Determines PART object A can be reconstructed from the primitives it shares with PART object B	The *TOPO_PART_IDs*	TRUE or FALSE
reconstruct_b	Determines PART object B can be reconstructed from the primitives it shares with PART object A	The *TOPO_PART_IDs*	TRUE or FALSE

Table 105 – Functions included in GEN_SHARED_ROUTINES_9I

STS - Package WHAT_RELATIONSHIP

A brief description of the functions contained within this package is given in Table 106 below.

Function Name	*Description*	*Parameters*	*Returns*
find_intersecting_objects	Returns the *FEATURE_IDs* of any objects intersecting with the specified object	*FEATURE_ID*	PL/SQL table listing intersecting objects and R-Code for the relationship
find_object_with_relationship	Returns the *FEATURE_IDs* of any objects having a specified relationship with the given object	*FEATURE_ID* R-Code for the required relationship	PL/SQL table listing the objects and the relationship

Table 106 - Functions included in WHAT_RELATIONSHIP

FIND_ INTERSECTING_ OBJECTS finds all Nodes associated with the object, (by querying the TOPO_NODE table) and then identifies other objects associated with any of these Nodes, using an inner join with the *TOPO_NODE* table. The results are returned as a pipelined PL/SQL table. In order to improve implementation performance, an assumption has been made that all objects are associated with Node primitives. If this is not the case, an additional three tests (on the Edge, Face and Volume primitives) must also be implemented here.

FIND_OBJECT_WITH_RELATIONSHIP finds all the intersecting objects as above, and iterates through the list, testing the relationship of each intersecting object with the object in question, using the code implemented in the NINE_INTERSECTIONS package. If the relationship matches the code required, then return the matching *FEATURE_ID*.

PL/SQL for this package is given here.

```
-- package for the following queries
-- 1.  which object is in a topological relationship with this object and what is the
relationship (r-code)
-- 2.  which object has relationship A with this one
create or replace type what_relationship_type as OBJECT (GEOMETRY_ID NUMBER(10),
RELATIONSHIP_TYPE NUMBER(10));
/
CREATE OR REPLACE TYPE what_relationship_table AS TABLE OF what_relationship_type;
/
CREATE OR REPLACE PACKAGE what_relationship AS
        FUNCTION find_intersecting_objects(geometry_id IN PLS_INTEGER) RETURN
what_relationship_table PIPELINED;
        FUNCTION find_object_with_relationship(geometry_id IN PLS_INTEGER,
relationship_code IN PLS_INTEGER) RETURN what_relationship_table PIPELINED;
END what_relationship;
/
CREATE OR REPLACE PACKAGE BODY what_relationship AS
        FUNCTION find_intersecting_objects(geometry_id IN PLS_INTEGER) RETURN
what_relationship_table PIPELINED
        IS
                sql_stmt VARCHAR2(1000);
                TYPE nodeCurTyp IS REF CURSOR;
                node_cv nodeCurTyp;
                node_rec TOPO_PART_TABLE.GEOMETRY_ID%TYPE;
                matching_geom_id PLS_INTEGER;
                relationship_result PLS_INTEGER;
        BEGIN
                -- first find all the geometries that have some relationship with A
                -- for each of these, identify the relationship
                sql_stmt := 'SELECT DISTINCT E.GEOMETRY_ID FROM TOPO_PART_TABLE E INNER
JOIN ';
                sql_stmt := sql_stmt || ' (SELECT D.PARENT_TOPO_ID FROM GEOM_NODE D INNER
JOIN ';
                sql_stmt := sql_stmt || ' (SELECT A.NODE_ID FROM GEOM_NODE A INNER JOIN
TOPO_PART_TABLE B';
                sql_stmt := sql_stmt || ' ON A.PARENT_TOPO_ID = B.PARENT_TOPO_ID WHERE';
                sql_stmt := SQL_STMT || ' B.GEOMETRY_ID = :1) C ON D.NODE_ID = C.NODE_ID)
F';
                sql_stmt := sql_stmt || ' ON F.PARENT_TOPO_ID = E.PARENT_TOPO_ID WHERE
E.GEOMETRY_ID != :2';

                OPEN node_cv FOR sql_stmt USING GEOMETRY_ID, GEOMETRY_ID;
                LOOP
                   FETCH node_cv INTO node_rec;
                        EXIT WHEN node_cv%NOTFOUND;
                        -- FIND THE RELATIONSHP AND STREAM OUT THE RESULT
                        matching_geom_id := node_rec;
                        -- find the relationship
                        sql_stmt := 'SELECT NINE_INTERSECTIONS.GET_GEOM_R_VALUE(:1,:2)
FROM DUAL';
                        EXECUTE IMMEDIATE sql_stmt INTO relationship_result USING
geometry_id, matching_geom_id;

                        pipe row
(what_relationship_type(matching_geom_id,relationship_result));
                END LOOP;
                CLOSE node_cv;
        END find_intersecting_objects;
```

```
        FUNCTION find_object_with_relationship(geometry_id IN PLS_INTEGER,
relationship_code IN PLS_INTEGER) RETURN what_relationship_table PIPELINED
        IS
                sql_stmt VARCHAR2(1000);
                TYPE nodeCurTyp IS REF CURSOR;
                node_cv nodeCurTyp;
                node_rec TOPO_PART_TABLE.GEOMETRY_ID%TYPE;
                matching_geom_id PLS_INTEGER;
                relationship_result PLS_INTEGER;
        BEGIN
                -- first find all the geometries that have some relationship with A
                -- for each of these, identify the relationship
                -- if the relationship is that required, add to the list and stream out!
                sql_stmt := 'SELECT DISTINCT E.GEOMETRY_ID FROM TOPO_PART_TABLE E INNER
JOIN ';
                sql_stmt := sql_stmt || ' (SELECT D.PARENT_TOPO_ID FROM GEOM_NODE D INNER
JOIN ';
                sql_stmt := sql_stmt || ' (SELECT A.NODE_ID FROM GEOM_NODE A INNER JOIN
TOPO_PART_TABLE B';
                sql_stmt := sql_stmt || ' ON A.PARENT_TOPO_ID = B.PARENT_TOPO_ID WHERE';
                sql_stmt := SQL_STMT || ' B.GEOMETRY_ID = :1) C ON D.NODE_ID = C.NODE_ID)
F';
                sql_stmt := sql_stmt || ' ON F.PARENT_TOPO_ID = E.PARENT_TOPO_ID WHERE
E.GEOMETRY_ID != :2';

                OPEN node_cv FOR sql_stmt USING GEOMETRY_ID, GEOMETRY_ID;
                LOOP
                   FETCH node_cv INTO node_rec;
                        EXIT WHEN node_cv%NOTFOUND;
                        -- FIND THE RELATIONSHP AND STREAM OUT THE RESULT
                        matching_geom_id := node_rec;
                        -- find the relationship
                        sql_stmt := 'SELECT NINE_INTERSECTIONS.GET_GEOM_R_VALUE(:1,:2)
FROM DUAL';
                        EXECUTE IMMEDIATE sql_stmt INTO relationship_result USING
geometry_id, matching_geom_id;

                        IF relationship_result = relationship_code THEN
                                pipe row
(what_relationship_type(matching_geom_id,relationship_result));
                        END IF;
                END LOOP;
                CLOSE node_cv;

        END find_object_with_relationship;
END what_relationship;
/
```

3DFDS - Package GEN_WHAT_RELATIONSHIP

A brief description of the functions contained within this package is given in Table 107 below.

Function Name	*Description*	*Parameters*	*Returns*
gen_find_intersecting_objects	Returns the *FEATURE_IDs* of any object intersecting with the specified object	*FEATURE_ID*	PL/SQL table listing intersecting objects and R-Code for the relationship
gen_find_object_with_relationship	Returns the *FEATURE_IDs* of any objects having a specified relationship with the given object	*FEATURE_ID* R-Code for the required relationship	PL/SQL table listing the objects and the relationship
gen_sql_stmt	Returns the combined SQL statement to identify any intersecting objects		VARCHAR2 containing the SQL statement

Table 107 - Functions included in GEN_WHAT_RELATIONSHIP

FIND_INTERSECTING_OBJECT first identifies any intersecting objects, then iterates through them checking for the specific relationship on a case-by-case basis. Any objects matching the relationship are returned as a pipelined PL/SQL table. However, the query to identify the intersecting objects references ALL the tables in the 3DFDS structure, as the dimension of potentially intersecting objects is not known. The query first identifies the Nodes associated with the given object by joining the *TOPO_NODE*, *TOPO_EDGE*, *TOPO_FACE* or *TOPO_VOLUME* tables through to the Node primitives. For example, to determine the Nodes associated with a 3D Body object, the *TOPO_VOLUME*, *FACE_VOLUME*, *EDGE_FACE* and *NODE_EDGE* tables must be joined. The reverse join must be followed to identify any intersecting 3D bodies. Additional joins must then be followed to identify intersecting 2D surfaces, 1D lines and 0D points. Finally the *NODE-IN-VOLUME* and *NODE-IN-FACE* exception tables must be queried to identify any additional intersections.

FIND_OBJECT_WITH_RELATIONSHIP finds all the intersecting objects as above, and iterates through the list, testing the relationship of each intersecting object with the object in question, using the code implemented in the GEN_NINE_INTERSECTIONS package. If the relationship matches the code required, then return the matching *FEATURE_ID*.

Package – TOPOLOGY

USER_QUERY_TEXT	*R-CODE*
INTERSECT	467
INTERSECT	511
INTERSECT	447
CONTAINS	220
INSIDE	220
INSIDE	179
NOT TOUCHING	31
TOUCHING	287
TOUCHING	317
IMPACTED	317
IMPACTED	316
IMPACTED	287
IMPACTED	285

Table 108 - Mapping User Terminology to R-Values (USER_QUERY_TO_R_CODE)

Table 108 illustrates a sample of the mapping process between user or domain-specific terminology and 9-Intersection R-Values.

Function QUERY queries the USER_QUERY_TO_R_CODE table to identify any R-Code values matching the specified user query text. Queries the FIND_INTERSECTING_OBJECTS code to identify candidate objects, and returns those whose relationship matches any of those in the list from the USER_QUERY_TO_R_CODE table.

Function QUERY_PAIR takes two *FEATURE_ID*s, and validates the R-Value of the relationship between them against the R-Values associated with the user query text. It returns the relationship between the objects using end-user terminology, by mapping the resulting R-Value to the user-terms specified.

Function QUERY_ANY takes a single *FEATURE_ID* and returns any non-disjoint objects (where a disjoint object is taken to have R-CODE 031).

Appendix 7 – SQL Tuning

This Appendix gives a brief description of the processing of a SQL query within an Oracle context, and of the TOAD software used to generate query execution plans.

Oracle and SQL

Running SQL Queries in Oracle

Once a query has been submitted to the Oracle database, a three stage process (Figure 128) is used to run the query and return the results to the user. The first of stage is the query parse process, where query optimisation occurs to ensure that the query is executed utilising minimum resources. The output of this process is an execution plan for the query, detailing the order in which tables are to be referenced and any indexes to be used. This is followed by query execution, in which the plan generated by the parsing phase is executed. Finally, data blocks are retrieved during the fetch phase, and resulting rows are returned to the calling application.

Step 1 PARSE	Step 1a QUERY TRANSFOR-MATION	Certain query constructs (views, subqueries) may be transformed (for example into joins) to open up new access possiblities	
	Step 1b COST BASED OPTIMISATION	Determine Object Cost and Cardinalities	Individual objects are costed and the number of rows (cardinality) is determined
		Cost Different Join Orders	Join orders and methods are evaluated and the plan with the lowest overall cost is chosen
		Build Structures for Runtime	Run time structures are built and stored in the library cache. At execution time, these are used to drive the query
Step 2 EXECUTE	Memory areas are allocated for bind variables, values are filled and the plan generated by the PARSE phase is executed		
Step 3 FETCH	Data blocks are retrieved, unwanted rows are removed and data is stored as necessary. Resultant rows are passed to the applicaiton		

Figure 128 - running a SQL query in Oracle – adapted from Oracle (2002)

The Oracle SQL Query Optimizer (Oracle 2006c) is responsible for the parsing of SQL queries within the Oracle database engine, determining the most efficient way to run the query. The optimizer is responsible for the evaluation of each SQL statement, generating a set of potential execution plans (including varying options for join order choice, options for join types, options for table scan processes) and then estimating the cost of each plan based on statistics available for the data – how the data is stored, whether it is stored on single or multiple disks. Computer resources – processor, Input/Output and memory are also taken into account as part of the evaluation process.

The optimization process in Oracle 10g is a cost-based optimisation – each operation required to process a query is assigned a cost in terms of these system resources and rows processed. The

total cost of each potential query execution plan is then calculated to allow the system to select the 'cheapest' plan.

Oracle's Explain Plan

Explain Plan information provided by Oracle details the outcome of the optimizer parsing process. The Explain Plan for a query lists the step by step process followed by the database software when executing that query. The results of running the explain plan process are presented in tabular format, with the following columns of the Explain Plan output table (Table 109) being relevant to non-parallelised queries.

Explain Plan Column	*Description*
Operation	This column details the type of SQL query being executed – i.e. whether it is a SELECT, INSERT, UPDATE or DELETE query. Additionally, information relating to sub-elements of the query is also given, including the type of join utilised, whether any index scans have occurred and whether any full table reads have been identified.
Object Name	This gives the name of the table or index being references by a particular step in the execution plan.
Rows	This gives an estimate of the total number of rows accessed by the query. If appropriate, the number is converted to K (thousands), M (millions), or G (1000 millions, or billions).
Bytes	This gives an estimate of the total number of bytes accessed by the query. If appropriate, the number is displayed in Kilobytes, Megabytes, or Gigabytes.
Cost	This is the value of the COST column of the plan table. If appropriate, the number is displayed in K (thousands), M (millions), or G (1000 millions, or billions). Cost is not determined for table access operations. The value of this column does not have any particular unit of measurement; it is merely a weighted value used to compare costs of execution plans.

Table 109 - Fields in the Explain Plan Table (from TOAD 2006)

Although all the fields listed above provide relevant information in terms of monitoring SQL performance, it is the OPERATION field that is of greatest relevance. This lists each operation (sort, search, index lookup, join) undertaken by the database engine to execute the given query. In particular, it also lists full table scans, which require the database to read (i.e. retrieve data from disk, store in memory and process) each and every record in a table, should be avoided where possible, particularly due to the higher cost of disk read operations. This becomes particularly relevant with the larger datasets under consideration as part of this research – tables may contain up to 13 million records.

A description of OPERATION values encountered during the SQL review process is given in Table 110 below for indices and in Table 111 for join operations. A join is characterised by multiple tables in the FROM clause and the relationship between the tables is defined through the existence of a JOIN condition in the WHERE clause. The query optimizer considers two factors when creating a plan for a join:

- Which of the join tables if any returns a single row of data (by referencing UNIQUE or PRIMARY KEY fields on the table). If this exists, this join is placed FIRST in the join order (i.e. the table is queried first and the results passed to the second part of the join process).
- If the join is an OUTER join, the outer join table is always queried second.

Further information can be found in Oracle (2006b).

Name	*Description*	*Used*
Full Table Scan	Reads all the rows from the table and filters out those that do not meet the selection criteria. Note that as Oracle performs I/O on blocks not rows, the decision to use a full table scan as opposed to an index is taken based on the number of blocks accessed.	When there is no index When the query is likely to require most of the data from a table When a table is small (i.e. smaller than DB_FILE_MULTIBLOCK_READ_COUNT * BLOCK_SIZE)
ROWID Scans (Table Access by Index ROWID)	The ROWID of a row specifies the data file and the block containing the row and the location of the row in that block. This is the fastest way to retrieve a single row.	This is generally the second step after retrieving the ROWID from an index, if the index itself does not contain all the columns needed for the query
Index Scans	A row is retrieved by traversing the index. If the columns required are present in the index itself, then the table is not queried. Various different index scan types can be identified	
Index Unique Scan	Returns a single ROWID.	Performed if a statement contains a UNIQUE or PRIMARY KEY constraint that guarantees that only a single row is accessed
Index Range Scan	Range can be bounded on both sides or unbounded. Index values are scanned in ascending order.	Used when one or more leading columns of an index are included in the SQL filter statement. Avoids sorting when index columns constitute ORDER BY/GROUP BY clause
Index Skip Scans	Retrieval of ROWIDs from a concatenated index without using the leading column(s) in the index. Improve index scans by non-prefix columns. Skip scanning allows a composite index to be split into smaller sub-indexes – the initial column of the composite index is skipped	Used with composite (multiple column) indexes.
Index Full Scan	Searches the entire index	Used if a predicate references one of the columns in the index
Fast Full Scan	Retrieval of all ROWIDs (and column values) using multiblock reads. No sorting order can be defined. Compares to a full table scan on only the indexed columns. Only available with the cost based optimizer.	Provides an alternative to a full table scan when the index contains all the columns necessary for the query

Table 110 – Operations in Explain Plan

Join Type	*Description*	*Where Used*
Nested Loop	Operation accepting two sets of rows, an outer set and an inner set. Oracle compares each row of the outer set with each row of the inner set, returning rows that satisfy a condition. For every row in the outer table, Oracle accesses all the rows in the inner table.	Useful when small subsets of data are being joined and if the join condition is an efficient way of accessing the second table
Hash Joins	The optimizer uses the smaller of the two tables to build a hash table on the join key in memory. It then scans the larger table (which may be on disk) accessing the records in the smaller table directly from memory.	Useful for joining large datasets or when a large fraction of a small table needs to be joined. Best when the smaller table fits into available memory.
Sort Merge Join	Used to join rows from two independent sources.	Used when the row sources are already sorted and no sort operation is required. Unless these conditions are met, use a HASH join as it generally performs better

Table 111 - Types of Join Operation

TOAD for Oracle Software

TOAD (TOAD 2006) provides a user-friendly graphical interface to analyse SQL statements and functionality to compare database schemas (including package code. It makes use of Oracle's inbuilt query Explain Plan functionality to provide guidelines on query optimisation, providing a graphical user interface to facilitate the generation of these plans. Further details can be found at TOAD (2006) and Scalzo (2006) provides details of automated 'best practices' testing.

SQL Execution Plans

Explain Plan results for the SQL queries run within each package are included in the attached CD for reference. In the SQL scripts, the TOPO_ tables described for both the STS and Extended 3DFDS structures in Chapter 4 and Chapter 5 are named GEOM_.

Appendix 8 – Databases and Database Configuration

This Appendix gives background information for the selection of an approach to extend standard SQL to support topological queries and ensure that 3D Topology functionality can be integrated into 3D GIS (Chapter 8). Background information relating to database theory and SQL is also overviewed, and details of the hardware and software selected for implementation are given.

Integrating Topology into 3D GIS – Object-Relational Databases

Date (1990) defines a database as a computerized record keeping system, while Atzeni *et al.* (1999) use the definition:

> *"A database is a collection of data used to represent information of interest to an information system".*

A database can be paper based or computerised; computerised databases are generally large, built for significant applications and shared amongst many users. A database management system (DBMS) is defined as a software system to manage the data stored in the database, and provides functionality such as reporting, backup facilities and security controls as well as the facility to enter data into the database, modify the data and interrogate (query) the data.

A Relational Database uses tables to store information. Object-Relational databases take this concept one step further, allowing the definition of objects, which can then be constructed from nested data types and tables. Tables or objects can then be linked to each other through a series of rules called constraints which restrict the type of data entered into the database and define the links between the various entities. Relational database usage has now become widely accepted in GIS, with commercial database packages such as Oracle (Oracle 2006a) and Informix (Informix 2007) offering specific functionality to support GIS requirements. Many GIS software packages also provide functionality to connect to such databases.

The advantages of a database and the associated management system include the integration of tools such as visualisation and data analysis, integration and centralisation of data from multiple sources, security, large numbers of concurrent users, seamless query within a single environment and centralised backup and recovery facilities. SQL (Structured Query Language) is also packaged with most relational databases, and is designed for the sole purpose of creating and querying data contained in a relational or Object-Relational database, hiding the complexity of the file system and structure of the database from the end user.

SQL is a set-based declarative language as commands are based on the set theory that underpins relational algebra (Atzeni *et al.* 1999) rather than procedures, although procedural elements have been added by some vendors. The SQL commands wrap the actual processes required to handle data within the database, which are also hidden from the user. SQL allows users to create tables, add rules, and insert, update or delete data in the tables, and also allows users to interrogate or query the data – i.e. to answer questions.

Existing Implementations

Given the context of an Object-Relational database, a review of existing database implementations of topology may provide guidance as to an appropriate mechanism to develop the required functionality. Two such implementations have been identified, namely Oracle 10g Topology (Oracle 2007b) and Radius Topology (Laser-Scan 2007). These are described briefly here.

Oracle 10g Topology supports topology management in the form of a topology structure which allows users to work with Node, Edge and Face primitives. The topological functionality is provided explicitly to support applications requiring data to be stored in a topological structure, in particular networking applications. Two separate topology types are offered – network and planar. Both implementations are totally separate from the geometry model provided by SDO_GEOMETRY and there is no direct link or mechanism to automatically populate the topological model from the SDO_GEOMETRY. Topology in 10g offers support for 2-dimensional data. The Node table contains an SDO_GEOMETRY object to support the point data representing the Node. The Edge table contains line geometry. The Face table does not, however, contain explicit geometry as such except for the minimum bounding rectangle (MBR) of the Face. Topology is edited via PL/SQL or Java functions.

Radius Topology, developed by 1Spatial (Laser-Scan 2007), offers 2 and 2.5D topological functionality in conjunction with Oracle Spatial and deals with both Node/Edge and planar topology (i.e. the Face can also be represented as a spatial object). The difference between this approach and that implemented by Oracle in 10g is the inbuilt link between the topology object and the original feature geometry, and the presence of automated functionality to populate and update the topology structure from the SDO_GEOMETRY representing the Feature. This allows the geometry to be used for visualisation tasks and allows topology to be utilised for data quality control and analysis issues. The automatic creation of the topology from the geometry also allows users to utilise existing data and build topology with this data rather than migrate the data into topological format. Topological primitives such as Nodes can be edited through

standard GIS interfaces, with automatic update of both connected Features and connected primitives.

Both Oracle 10g Topology and Radius Topology offer integrated query functionality to determine, for example, if two Features share a Node, to identify connecting links in a network, or to reconstruct a Feature from its constituent primitives. This is presented as a number of extensions to the standard SQL offered by the core Oracle database.

Integrating Topology into 3D GIS – Extending SQL

Although this research focuses specifically on the topological relationships between two features user requirements are rarely expressed in a purely topological form. In fact, end-user queries involving binary topological relationships are generally mixed, also requiring the determination of other spatial relationships such as direction or distance and the identification of non-spatial attributes of particular features. Thus an implementation approach is required that facilitates integration of topology with other query types.

The SQL language can be extended in a number of ways. The first of these is by extending the core syntax of the language to include new functionality. This is done with each new release of SQL and involves the collaboration of industry vendors, developers and other interested organisations feeding into the standards definition process.

The second method is to take advantage of the procedural extensions provided by SQL implementers to define a number of custom procedures that users and developers can then call to obtain the required analytical functionality. These procedures will include calls to standard SQL which in turn interrogates the data stored in the database. These procedural extensions are offered by the large majority of Object-Relational database implementations.

Additionally, SQL commands can also be issued from other programming languages such as Visual Basic, C, C++ and Java, allowing information from the database to be processed externally to the database management system. These languages can be used to wrap the complexity of SQL into interfaces more suitable for non-expert end-users.

Extending Oracle SQL - PL/SQL

PL/SQL (Procedural Language/SQL) is a procedural language extension within the Oracle DBMS that allows the development of programs in the SQL environment within the database itself, providing a tighter link to the database engine than other development languages such as Java or Visual Basic. Additionally, PL/SQL directly accesses native Oracle data types,

including the Object-Relational types such as SDO_GEOMETRY, which removes the requirement to cast data into programming language variable types before operations are performed. This makes PL/SQL an ideal programming language where intensive SQL querying is required.

In the case of the work described in this thesis, the code simply acts as a facility to group a sequence of SQL queries. Given the dominance of these SQL queries when compared to other code elements (loops, conditional clauses, mathematical algorithms), PL/SQL was selected for implementation. The choice of implementation language would, of course, need to be reconsidered should more computationally intensive algorithms be required.

The basic construct of a PL/SQL program is a block, with variable declarations in the header, and code delimited with a BEGIN and END tag. Blocks are in turn stored within procedures, which can be called directly from the SQL command prompt, from within other PL/SQL blocks, within SQL statements or from third-party programming languages. Similar procedures can, in turn, be grouped into packages (Atzeni *et al.* 1999). Standard programming constructs such as variables, loops and conditional statements are available within PL/SQL. Additionally, the concept of a CURSOR is used to iterate through results generated from SQL queries. PL/SQL tables can also be used to allow procedures to return query results in a similar way to standard queries – the results are presented to the user as a virtual table. To improve performance, bind variables have been used for any standard SQL embedded in the PL/SQL block. These allow SQL queries to be pre-interpreted and stored in memory rather than interpreted as they are encountered – the query is only interpreted the first time it is encountered and the interpreted version used every subsequent time (see Appendix 7 for details of query parsing in Oracle).

Hardware Configuration

Brief details of the system hardware and key Oracle database configuration parameters used for the work described in this thesis are given here.

Database Server

Parameter	*Value*
Operating System	Microsoft Windows 2003, Enterprise Edition, Service Pack 1
Processors	4 * Intel Xeon CPU 3.06 GHz
RAM	8 GB
Disks	5 * Seagate SCSI Drives ST336607LC
Total available storage	1 * 34.1 GB, 4 * 68.3 GB

Table 112 - Database Server Configuration

Test Hardware

Parameter	*Value*
Operating System	Microsoft Windows XP Professional SP 2 Version 2002
Processor	Intel Pentium 4 Processor, 3.0GHz, Single CPU
RAM	1GB

Table 113 – Test Hardware Configuration

Oracle Database Configuration

Parameter	*Value*
Database Version	Oracle Database Server: 10.1.0.3.0 Enterprise Edition – Production, with Partitioning, OLAP and Data Mining Options
SQL Plus	SQL*Plus: Release 10.1.0.3.0
Total Database Size	251833.75 MB (for all users, includes Ordnance Survey MasterMap topographic mapping for the London area)

Table 114 – Oracle Database Configuration

Oracle Database Initialisation Parameters

Parameter	*Value*	*Description*
DB_BLOCK_SIZE	16 MB	The size (in bytes) of an Oracle database block.
DB FILE MULTI-BLOCK READ COUNT	16	The maximum number of blocks read during an I/O operation involving a full sequential scan.
OPTIMIZER MODE	ALL_ROWS	Specifies the behaviour of the optimizer. ALL_ROWS uses cost-based optimisation unless query hints are included in the SQL statement.
PGA AGGREGATE TARGET	1200 MB	The target aggregate PGA (program global area) memory for all server processes attached to the instance. The PGA is used to process SQL statements and to hold logon and other session information.
SGA MAX SIZE	1508 MB	The maximum size of the System Global Area for the lifetime of the instance. The SGA is a shared memory area that contains data and control information for the instance. Multiple users can share data within this memory area and avoid repeated, time-consuming access from physical disk.
SHARED POOL SIZE	800 MB	Specifies the size of the shared pool. The shared pool contains objects such as shared cursors, stored procedures, control structures, and Parallel Execution message buffers. Larger values can improve performance in multi-user systems.
SORT AREA SIZE	65536 KB	Specifies the maximum amount, of memory to use for a sort. After the sort completes, rows are returned and the memory is released. Temporary disk segments are used if memory is exceeded.

Table 115 – Oracle Database Initialisation Parameters (Oracle 2006f)

Appendix 9 – Additional Test Results – Preliminary Tests

This Appendix contains the numerical data underpinning the graphs shown in Chapter 9 (Performance Test Results). Test results are given for three of the four preliminary tests, namely:

- R-Tree Index Tolerance Value
- Random or Sequential Feature ID Generation
- Number of Test Iterations

Test results for the main tests are given in Appendix 10. Tests results do not include the additional time included added to the Proxy for 'As-Required' queries. As these additional time values are constant, they are less relevant to these preliminary tests.

Preliminary Test 1– Random or Sequential FEATURE_ID Generation

Ran./Seq.	*Users*	*Test Name*	*Min (s)*	*Max (s)*	*Avg (s)*	*StdDev*
RANDOM	1	*9-Intersection Pairs*	0.05858	0.0697	0.06313	0.00583
RANDOM	2	*9-Intersection Pairs*	0.07688	0.09641	0.08532	0.00733
RANDOM	4	*9-Intersection Pairs*	0.14938	0.17423	0.1624	0.00678
RANDOM	6	*9-Intersection Pairs*	0.18719	0.27549	0.23349	0.02799
RANDOM	8	*9-Intersection Pairs*	0.24017	0.35424	0.31788	0.02973
SEQ	1	*9-Intersection Pairs*	*0.03078*	*0.03782*	*0.03484*	0.00364
SEQ	2	*9-Intersection Pairs*	0.04641	0.05391	0.05128	0.00273
SEQ	4	*9-Intersection Pairs*	0.07251	0.10156	0.0887	0.00849
SEQ	6	*9-Intersection Pairs*	0.07703	0.17501	0.14129	0.02974
SEQ	8	*9-Intersection Pairs*	0.14126	0.22907	0.19134	0.02755

Table 116 – STS, Random and Sequential Object IDs, 135168 Objects

Ran./Seq.	*Users*	*Test Name*	*Min (s)*	*Max (s)*	*Avg (s)*	*StdDev*
RANDOM	1	*9-Intersection Pairs*	0.46674	0.51237	0.48914	0.02283
RANDOM	2	*9-Intersection Pairs*	0.57768	0.79332	0.68938	0.07962
RANDOM	4	*9-Intersection Pairs*	0.95301	1.12583	1.04599	0.05643
RANDOM	6	*9-Intersection Pairs*	1.40741	2.16104	1.65033	0.1956
RANDOM	8	*9-Intersection Pairs*	1.94993	3.39015	2.35703	0.3465
SEQ	1	*9-Intersection Pairs*	0.30861	0.31955	0.31517	0.00579
SEQ	2	*9-Intersection Pairs*	0.32079	0.52612	0.44523	0.07191
SEQ	4	*9-Intersection Pairs*	0.49784	0.61058	0.56636	0.03974
SEQ	6	*9-Intersection Pairs*	0.78472	1.2249	1.0001	0.11178
SEQ	8	*9-Intersection Pairs*	1.19054	1.68584	1.43949	0.12727

Table 117 - Random and Sequential IDs, 3DFDS, *9-Intersection Pairs*

Ran./Seq.	*Users*	*Test Name*	*Min (s)*	*Max (s)*	*Avg (s)*	*StdDev*
RANDOM	1	Proxy for *9-Intersection Pairs*	0.05687	0.08266	0.025887	0.01459
RANDOM	2	Proxy for *9-Intersection Pairs*	0.08751	0.09109	0.068643	0.00148
RANDOM	4	Proxy for *9-Intersection Pairs*	0.07828	0.10719	0.062387	0.00819
RANDOM	6	Proxy for *9-Intersection Pairs*	0.05313	0.13282	0.061774	0.01732
RANDOM	8	Proxy for *9-Intersection Pairs*	0.06172	0.18985	0.067068	0.02493
SEQ	1	Proxy for *9-Intersection Pairs*	0.06625	0.07891	0.037973	0.01097
SEQ	2	Proxy for *9-Intersection Pairs*	0.23384	0.26063	0.069137	0.00584
SEQ	4	Proxy for *9-Intersection Pairs*	0.19735	0.24798	0.06189	0.0061
SEQ	6	Proxy for *9-Intersection Pairs*	0.22829	0.25329	0.060432	0.00807
SEQ	8	Proxy for *9-Intersection Pairs*	0.12626	0.27517	0.070824	0.03653

Table 118 - Random and Sequential IDs, Proxy for As-Required Structure

Preliminary Test 2– Number of Test Iterations

Note that a single run of each test was executed here.

Number of Iterations	*Random or Sequential*	*Test Name*	*Test Time (s)*
10	RANDOM	*9-Intersection Pairs*	0.15000
100	RANDOM	*9-Intersection Pairs*	0.09516
500	RANDOM	*9-Intersection Pairs*	0.07375
1000	RANDOM	*9-Intersection Pairs*	0.05486
5000	RANDOM	*9-Intersection Pairs*	0.04953
10000	RANDOM	*9-Intersection Pairs*	0.05106

Table 119 – Data for STS *9-Intersection Pairs*, varying Iterations, 135168 Objects

Number of Iterations	*Random or Sequential*	*Test Name*	*Test Time (s)*
10	RANDOM	*9-Intersection Pairs*	*0.03100*
100	RANDOM	*9-Intersection Pairs*	0.27500
500	RANDOM	*9-Intersection Pairs*	0.06281
1000	RANDOM	*9-Intersection Pairs*	0.03856
5000	RANDOM	*9-Intersection Pairs*	0.03588
10000	RANDOM	*9-Intersection Pairs*	0.03465

Table 120 – Data for As-Required Proxy, varying Iterations, 135168 Objects

Number of Iterations	*Random or Sequential*	*Test Name*	*Test Time (s)*
10	RANDOM	*9-Intersection Pairs*	1.72970
100	RANDOM	*9-Intersection Pairs*	1.62768
500	RANDOM	*9-Intersection Pairs*	1.59005
1000	RANDOM	*9-Intersection Pairs*	1.73221
5000	RANDOM	*9-Intersection Pairs*	1.64866
10000	RANDOM	*9-Intersection Pairs*	1.59570

Table 121 – Data for 3DFDS *9-Intersection Pairs*, varying Iterations, 135168 Objects

Preliminary Test 3 – R-Tree Index Tolerance Value

As Required – 9-Intersection Pairs – 1.08 Million Objects

Tolerance (m)	*Users*	*Test Name*	*Maximum (s)*	*Minimum (s)*	*Average (s)*	*Standard Deviation*
0.05	1	Proxy for *9-Intersection Pairs*	*0.07297*	*0.10079*	*0.08225*	0.01606
0.5	1	Proxy for *9-Intersection Pairs*	0.07048	0.08532	0.076	0.00812
1	1	Proxy for *9-Intersection Pairs*	0.07063	0.08376	0.07777	0.00664
5	1	Proxy for *9-Intersection Pairs*	0.09938	0.11907	0.10777	0.01016
100	1	Proxy for *9-Intersection Pairs*	0.10985	0.12892	0.11751	0.01007
500	1	Proxy for *9-Intersection Pairs*	0.30767	0.31595	0.31298	0.00461

Table 122 - As Required Queries, 1.08 million Objects, Random IDs, varying Tolerance

Preliminary Test 4 – Improving the Proxy for As-Required Queries

No additional test results.

Appendix 10 – Additional Test Results – Main Tests

This Appendix contains the summarised data underpinning the graphs shown in Chapter 9 (Performance Test Results). Raw data for these tests is available on the attached CD. The results shown here include values added to improve the As-Required Proxy.

9-Intersection Pairs – Proxy for As-Required

Number of Users	*Time for 264 Features (seconds)*	*Time for 2112 Features (seconds)*	*Time for 16896 Features (seconds)*	*Time for 138168 Features (seconds)*	*Time for 1081344 Features (seconds)*
1	3.040861	3.183404	2.979611	2.508713	3.8488
2	1.968401	1.9222s	1.967276	2.357421	4.746743
4	0.902907	0.83341s	0.911326	1.199893	2.529418
6	0.674394	0.692866s	0.747665	0.831944	1.67067
8	0.479808	0.464742s	0.508646	0.627746	1.1329

Table 123 – Proxy for As-Required Scalability Test Results –*9-Intersection Pairs*

9-Intersection Pairs – STS

Number of Users	*Time for 264 Features (seconds)*	*Time for 2112 Features (seconds)*	*Time for 16896 Features (seconds)*	*Time for 138168 Features (seconds)*	*Time for 1081344 Features (seconds)*
1	0.04552	0.043383	0.047083	0.063127	0.054167
2	0.070783	0.072943	0.074767	0.085315	0.082893
4	0.158508	0.172024	0.164018	0.162403	0.152885
6	0.21575	0.21105	0.203458	0.233494	0.236433
8	0.305095	0.318783	0.307649	0.317879	0.34134

Table 124 - Scalability Tests for STS, *9-Intersection Pairs*

9-Intersection Pairs – 3DFDS

Number of Users	*Time for 264 Features (seconds)*	*Time for 2112 Features (seconds)*	*Time for 16896 Features (seconds)*	*Time for 138168 Features (seconds)*	*Time for 1081344 Features (seconds)*
1	0.5493	0.60023	1.278913	0.48914	1.13844
2	0.77909	1.025213	2.334592	0.689378	1.779943
4	1.284193	1.347918	2.972846	1.045994	2.15621
6	1.912976	2.086104	3.880936	1.650325	3.038692
8	2.639195	3.007321	5.337374	2.357026	3.867934

Table 125 - *9-Intersection Pairs* Scalability Test Results, 3DFDS

Find Intersecting Objects – Proxy for As-Required

Number of Users	*Time for 264 Features (seconds)*	*Time for 2112 Features (seconds)*	*Time for 16896 Features (seconds)*	*Time for 138168 Features (seconds)*	*Time for 1081344 Features (seconds)*
1	2.318928	2.188072	2.56771	2.504411	5.672399
2	1.489401	1.346151	1.782763	2.361327	6.403715
4	1.362805	1.133852	1.498704	1.7263	3.094593
6	0.99381	0.937326	1.265366	1.26851	2.057604
8	1.087264	1.165894	1.360261	1.437169	1.818363

Table 126 – As- Required Scalability Test Results – Find Intersecting Objects

Find Intersecting Objects - STS

Number of Users	*Time for 264 Features (seconds)*	*Time for 2112 Features (seconds)*	*Time for 16896 Features (seconds)*	*Time for 138168 Features (seconds)*	*Time for 1081344 Features (seconds)*
1	0.02823	0.029847	0.02651	0.040053	0.03198
2	0.044582	0.049713	0.041905	0.053465	0.04414
4	0.051095	0.062283	0.051759	0.068637	0.084978
6	0.072606	0.077712	0.062555	0.096802	0.123806
8	0.068355	0.06592	0.060145	0.087451	0.172714

Table 127 – STS Scalability Test Results – Find Intersecting Objects

Find Intersecting Objects – 3DFDS

Number of Users	*Time for 264 Features (seconds)*	*Time for 2112 Features (seconds)*	*Time for 16896 Features (seconds)*	*Time for 138168 Features (seconds)*	*Time for 1081344 Features (seconds)*
1	1.44793	4.495987	1.085467	0.3571	0.84738
2	2.09356	7.463153	1.34529	0.534948	1.308462
4	3.09089	10.06511	2.279099	0.715173	1.694568
6	5.120014	14.90428	2.903081	1.279599	2.493619
8	7.542625	19.92034	3.684462	1.655899	2.754576

Table 128 – 3DFDS – Find Intersecting Objects

Find Objects with Relationships – Proxy for As-Required

Number of Users	*Time for 264 Features (seconds)*	*Time for 2112 Features (seconds)*	*Time for 16896 Features (seconds)*	*Time for 138168 Features (seconds)*	*Time for 1081344 Features (seconds)*
1	18.03588	16.98463	17.77057	10.94838	19.74231
2	12.80858	10.62755	11.27162	9.052945	15.508
4	7.267715	5.97953	6.974011	6.919727	8.550988
6	7.154725	6.578522	7.803669	5.894435	5.342325
8	6.220212	5.796787	6.376818	4.832041	4.119875

Table 129 – Proxy for As-Required Scalability, Find Objects with Relationships

Find Objects with Relationships – STS

Number of Users	*Time for 264 Features (seconds)*	*Time for 2112 Features (seconds)*	*Time for 16896 Features (seconds)*	*Time for 138168 Features (seconds)*	*Time for 1081344 Features (seconds)*
1	0.0288	0.030573	0.029377	0.050627	0.032183
2	0.040627	0.049013	0.046903	0.064092	0.0475
4	0.072045	0.087731	0.076218	0.082728	0.083843
6	0.073536	0.080081	0.068317	0.097849	0.13266
8	0.084931	0.091573	0.08402	0.11996	0.18642

Table 130 – STS Scalability Test Results, Find Objects with Relationships

Find Objects with Relationships – 3DFDS

Number of Users	*Time for 264 Features (seconds)*	*Time for 2112 Features (seconds)*	*Time for 16896 Features (seconds)*	*Time for 138168 Features (seconds)*	*Time for 1081344 Features (seconds)*
1	1.38078	4.036763	1.14162	0.411777	0.907727
2	2.170175	6.305913	1.436698	0.519058	1.358965
4	3.178216	9.906302	2.197249	0.676807	1.638352
6	5.237394	14.73462	2.900816	1.276498	2.493844
8	7.375174	20.36451	3.637993	1.682797	2.879998

Table 131 – 3DFDS Scalability Test Results, Find Objects with Relationships

Appendix 11 – Contents of Attached CD

	Contents	Location
1	Screenshots of the 264 Objects forming the core dataset, showing associated binary topological relationships	..\dataset_diagrams\ Resulting_dataset_screenshots.pdf
2	Oracle export of the core dataset – Extended 3DFDS	..\exported_data_264_objects\3dfds\3dfds.dmp
3	Oracle export of the core dataset – As-Required Structure	..\exported_data_264_objects\as-required\as-required.dmp
4	Oracle Export of the core dataset - Simplified Topological Structure	..\exported_data_264_objects\sts\sts.dmp
5	PL/SQL code files – *9-Intersection Pairs* tests for 3DFDS	..\pl_sql_code\3DFDS
6	PL/SQL code files – all tests, As-Required Structure	..\pl_sql_code\As-Required
7	PL/SQL code files used to support the data structure population and replication processes	..\pl_sql_code\data_structure_population_and_test
8	PL/SQL code files – map the relationship R-Code to end user terminology	..\pl_sql_code\map_r_code_to_user_queries
9	PL/SQL code files – *Find Intersecting Objects* and *Find Objects with Relationships* tests, STS and 3DFDS	..\pl_sql_code\relationship_with_a_queries
10	PL/SQL code files – *9-Intersection Pairs* tests for STS	..\pl_sql_code\STS
11	SQL Explain Plan Results	..\sql_explain_plans\sql_tuning_explain_plan_output.pdf
12	Spreadsheet of raw test results each of the three tests run against each structure. Summary and comparative data also included	..\test_results\raw_test_results.xls